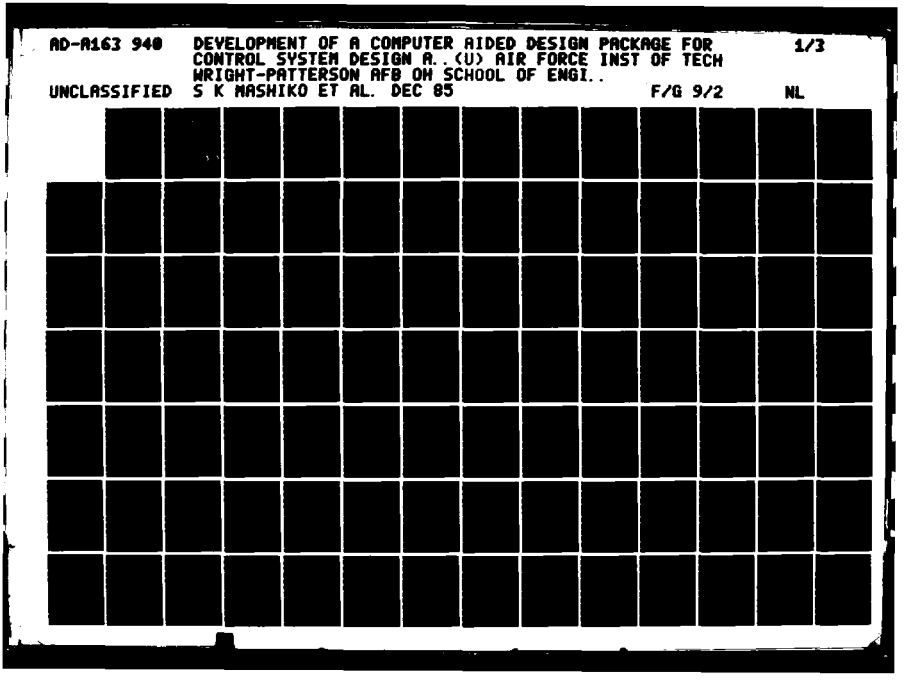
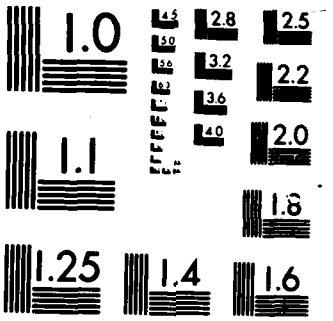

AD-A163 940 DEVELOPMENT OF A COMPUTER AIDED DESIGN PACKAGE FOR CONTROL SYSTEM DESIGN A.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..
1/3
UNCLASSIFIED S K MASHIKO ET AL. DEC 85
F/G 9/2
NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DEVELOPMENT OF A COMPUTER AIDED DESIGN PACKAGE FOR CONTROL SYSTEM DESIGN AND ANALYSIS FOR A PERSONAL COMPUTER (ICECAP-PC)

THESIS

Volume I of II

Susan K. Mashiko
Captain, USAF

Gary C. Tarczynski
Captain, USAF

AFIT/GE/ENG/85D-46

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/85D-46

DEVELOPMENT OF A COMPUTER AIDED DESIGN PACKAGE FOR CONTROL SYSTEM DESIGN AND ANALYSIS FOR A PERSONAL COMPUTER (ICECAP-PC)

THESIS

Volume I of II

Susan K. Mashiko
Captain, USAF

Gary C. Tarczynski
Captain, USAF

AFIT/GE/ENG/85D-46

Approved for public release; distribution unlimited

AFIT/GE/ENG/85D-46

DEVELOPMENT OF A COMPUTER AIDED DESIGN PACKAGE FOR CONTROL SYSTEM DESIGN AND ANALYSIS FOR A PERSONAL COMPUTER (ICECAP-PC)

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Susan K. Mashiko, B.S.A.E.
Captain, USAF

Gary C. Tarczynski, B.S.E.E.
Captain, USAF

December 1985

Preface

The purpose of this study was to develop a computer-aided design (CAD) package for control systems engineers. The main thrust of the effort was involved in implementing the program on a personal computer. Although the program could be run on larger mainframe computers, it was our intention to make this program useable on much smaller systems. We wanted to provide the control systems engineer with powerful design and analysis tools at his desk.

It is hoped that future students will build upon the program, and continue to expand its design and analysis capabilities. Throughout our thesis effort, we learned how difficult and time-consuming an interactive software project could be. Yet we also learned how rewarding and satisfying the experience is when the final product can perform its function swiftly, accurately, and consistently.

There were many people who were involved in our thesis effort in one way or another. We would like to thank our advisor, Dr. Gary Lamont, for giving us the freedom to pursue a topic of our own choosing. We would also like to thank our readers, Dr. Robert Fontana, Dr. John D'Azzo, and Capt. Steve Rogers. We would especially like to thank Gary's brother, Mark, who was at the receiving end of frantic phone calls to New York. His searches through the IBM database helped solve some of our thorniest problems. Special thanks goes to Capt. Ken Cole, for providing answers and suggestions that were always right and always helpful. A final thank you goes to Capt. John Bullard, a fellow student, a bright engineer, and a dear friend.

Table of Contents

List of Figures

List of Tables

Abstract

This ~~investigation developed~~ a computer-aided design (CAD) package for control system design and analysis. The package was implemented on different varieties of small personal computers.

Structured design and other software engineering techniques were applied during the development effort. The program consists of a keyword-driven menu structure and a set of control system analysis procedures. The analysis procedures allow input of systems which are defined by transfer functions, polynomials, and matrices. Polynomials and matrices can be manipulated mathematically, and some block diagram manipulation can performed on transfer functions as well.

The program is only part of a continuing development effort in the Information Sciences Laboratory at the Air Force Institute of Technology.

DEVELOPMENT OF A COMPUTER AIDED DESIGN PACKAGE FOR CONTROL SYSTEM DESIGN AND ANALYSIS FOR A PERSONAL COMPUTER

1. Introduction

Control system design and analysis involves a considerable amount of repetitive mathmatical manipulation including, but not limited to, polynomial factoring, matrix algebra, calculus, linear and non-linear differential equations, stochastic equations. With the advent of digital computers, many programs were written to aid the control engineer but each program only solved a specific problem (19). Ideally, the controls engineer requires a single integrated CAD package that can analyze all aspects of the design problem. Of particular interest, are the continuing efforts to develop an integrated controls package at the Air Force Institute of Technology (AFIT) at Wright-Patterson Air Force Base, Ohio.

1.1 Background

Under the guidance of Dr Gary Lamont, an AFIT effort to consolidate these specialized programs into a single design package was begun in 1977 by Fredrick L. O'Brian in his MS Thesis Consolidated Computer Program for Control System Design (17). In a follow-on thesis effort, by Stanley Larimer, the computer-aided design (CAD) program An Interactive Computer-Aided Design Program for Discrete and Continuous Control System Analysis and Synthesis was created (13). This program has the capability to analyze both discrete and continuous control systems. This program is called 'TOTAL' and was initially implemented in the higher order language FORTRAN on a Control Data Corporation CYBER computer. In 1981,

efforts were begun to move the package to a more common and widely available type of computer. Glen Logan rehosted TOTAL for the large CYBER mainframe onto a smaller Digital Equipment Corporation VAX-11/780 minicomputer (15). This version of TOTAL is known as VAXTOTAL. Logan also coined the term 'ICECAP', which stands for 'Interactive Control Engineering Computer Analysis Package'. This name became synonymous with the VAX version of TOTAL, and was to remain the controls package throughout all stages of its development.

Work continued on ICECAP, and in 1982, Charles Gembarowski added a user friendly interface (10). This menu driven interface (which is implemented in Pascal) made it easier for the control engineer to input, output, and analyze data. Development proceeded, and more improvements on ICECAP soon followed. In 1983, Robert Wilson (26) added further capabilities to the controls routines, while Mark Travis (24) extended the graphics interface. During 1984, Abraham Armold provided discrete controller design capabilites (1), and Chiewcharn Narathong added tools for matrix manipulation and modern control theory analysis (18). Work continues on improving both TOTAL and ICECAP.

TOTAL and ICECAP are currently used by students for control system analysis and design. In order to use either of these programs one has to have access to either the ASD CYBER for TOTAL, the AFIT/ENG Information Systems Laboratory VAX-780 or the AFIT/EN VAX 11/780 in Room 130 for ICECAP. It is not always possible nor convenient to access these computers. The CYBER has a limited number of dial-up lines and a large user population that slows the response time. The EN VAX has limited remote access capabilities. Because of the direct connect limitation the user population is by neccessity small. Even if ICECAP were to be

rehosted on a computer with remote access capabilities there would be problems with many of the features of the program. The home terminal is a line oriented device, and as such the special graphical capabilities of ICECAP would not function correctly (20). Rehosting TOTAL with modifications or even purchasing a commercially available CAD packages such as MATRIXx (21) would not solve the problems of the students' limited access or of the engineer who does not have access to a computer capable of running either TOTAL or ICECAP. A new solution for these problems must be found.

The problems of availability and access could be mitigated by developing a control system design package similar to ICECAP for the small personal computers that have become so popular in recent years. This type of CAD program provides the student and the engineer with their own design package. This package could provide a user friendly environment and graphical capabilities tailored to the particular personal computer or computer operating system. The capabilities of this program could easily be increased if modular techniques are employed in its design. Work in the area of providing a CAD package for control engineers using microcomputers was begun in 1983 by Vincent Parisi. He created a interactive menu driven interface, called Control-CAD or muTrol >>>>, that prompts the user through the use of command/key words (20). This effort stopped short of providing any fundamental control analysis tools. The menu system was further developed and improved by Paul Moore, in 1984 (17).

1.2 Purpose

The purpose of this thesis effort is to develop a computer-aided

design program for control system design and analysis to run on a microprocessor based machine, with initial emphasis on the International Business Machines IBM-PC and the Zenith Data Systems Corporation Z-100 personal computers and the Microsoft's MS-DOS operating system. This package shall be portable to other computers and to other operating systems. This package shall be user friendly and shall meet minimum accuracy requirements. Additionally, this program shall be public domain and by design can be easily modified or augmented with new subroutines.

1.3 Scope - Limits of the Problem

The intent is to modify the MICROSDW program developed by P. A. Moore in his thesis for use on an IBM-PC and the Z-100 with the PC-DOS operating system (17). Utilizing the modified MICROSDW create a menu driven shell or executive program that will act as an interface between the user and the subroutine modules. This shell will accept commands from the user, determine if the commands are valid, decode the command, and call the desired subroutine. This is illustrated in Figure 1.1

Rehost or develop subroutines for priority one modules. The priority one modules are outlined in the approach section of this Chapter. The entire list of priority one modules is considered necessary for program operation and basic control design and analysis. These subroutines modules shall accept and decode the user command, accept the user input, provide on-line assistance, and perform the desired mathmatical manipulation. A detailed description of the requirements for both the executive and the subroutines is provided in Chapter II.

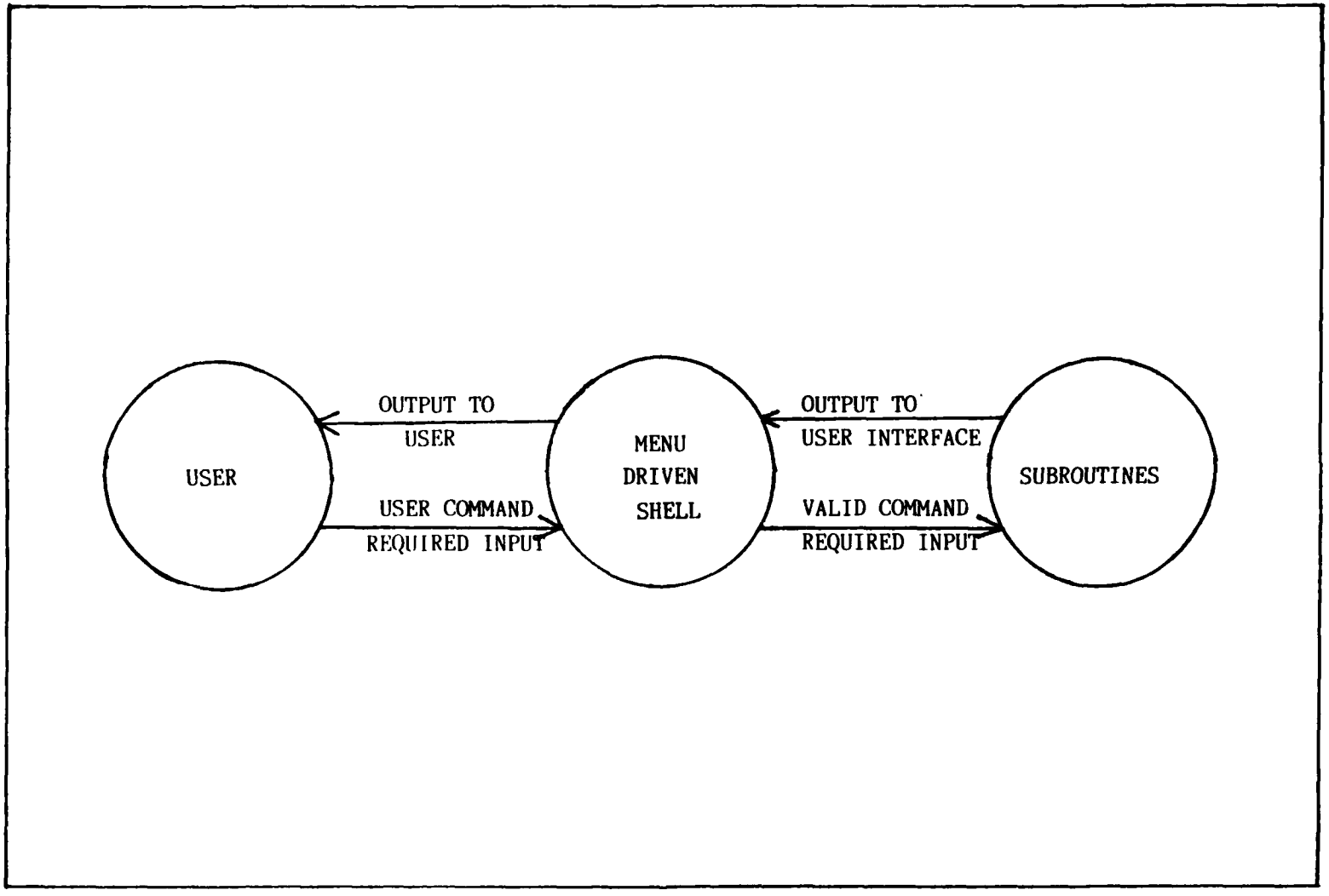


FIGURE 1.1: Menu Driven Shell/Executive

1.4 Summary of Current Knowledge

Currently, the computer program Control-CAD or muTrol>>>> is available. It does the basics of control design and analysis for use on a CP/M operating system based personal computer. This program has subroutines for polynomial input and root locus analysis. The root locus analysis is limited to poles and zeros on the real or imaginary axes. This program does not include all of the subroutines necessary for control system design and analysis (20). Commercially available programs for control system design and analysis are available, but at average of $350.00 or more the cost is prohibitive. In addition to the AFIT developed software and the commercial software, the Air Force's Aeronautical Systems Division is developing a control system design and analysis program in Basic for the Zenith 100 computer (23). A comprehensive list of other packages may be found in reference (2).

Current micro-computers have a limited memory capability and operate at a relatively slow speed. All design efforts for this thesis effort must be optimized for memory utilization and thru-put.

1.5 Standards

The program must be user friendly. The program should be easy to learn, easy to use, and should provide any necessary on-line assistance.

This program must provide results consistant with those from either TOTAL or ICECAP to the degree of accuracy available from the design language selected.

This program must be portable. This means that the program can be operated on a variety of personal computers and a variety of operating systems.

1.6 Approach

The system design effort is divided into three parts as follows: requirements analysis and resource selection, shell development, and sub-routine development.

During the requirements analysis and resource selection phase, study the requirements of interactive CAD programs in general and investigate similar CAD programs by means of a literature search. Determine the operating system under which this program shall be developed. Evaluate the various design languages available for speed, numerical capabilities, and their transportability. Prioritize the functions/sub-routines. The sub-routines that are required for early operational capability shall be priority one.

During the shell development phase modifications will be made to the MICROSDW program (17). These modifications are required to make this program compatible with a new operating system and to increase the user friendliness of MICROSDW. The modified MICROSDW will then be used to generate the shell. The combined package of the modified MICROSDW and subroutines shall be called 'ICECAP-PC', and hereafter shall be referred as such.

Subroutine development is the design and implementation of the modules that actually accomplish the numerical analysis. These sub-routines shall be called by the shell. These subroutines shall only be accessible to the user through the shell. The goal of this development is speed and accuracy.

Throughout this design effort the CAD package will be tested by the authors and by potential users for user friendliness, speed, and

accuracy. In addition, a complete documentation package will be created utilizing software engineering principles and techniques for each of the three parts of the design effort (7,14).

1.7 Sequence of Presentation

This thesis is divided into six different chapters and six appendicies. The combined package reflects the six phases of Software Development Life Cycle and a conclusion. The six phase of the life cycle are: System Planning Phase, Requirements Definition and Analysis Phase, Design Phase, Implementation Phase, Testing Phase, and Operations and Maintenance Phase (14). The following is a brief summary of the chapters and appendices presented in this thesis.

Chapter I contains the initial plan for the ICECAP-PC effort.

Chapter II contains the system requirements definition for ICECAP-PC. This thesis designs and implements a subset of these requirements.

Chapter III contains the preliminary design of ICECAP-PC. This chapter focuses primarily on the user interface and the selection of the resources for this development program. This chapter, along with the structure charts and the data dictionary make up a the Preliminary Design Document for this effort.

Chapter IV contains the details of the actual implemented design. Each of the options available in ICECAP-PC is discussed.

Chapter V contains the highlights of the ICECAP-PC development tests as well as the highlights of the validation tests run against TOTAL and ICECAP-PC.

Chapter VI contains the conclusions and the recommendations resulting from this thesis effort.

Appendix A is a User's Manual for ICECAP-PC and Appendix B is the Programmer's Manual. Instructions for the operation and the maintanence of ICECAP-PC can be found in these two documents. Appendix C is the structure charts for ICECAP-PC's shell and subroutines. Appendix D is the data dictionary for the same software. Listing of the source code may be found in Appendix E and copies of the text files for the BUILDDAT program may be found in Appendix F.

CHAPTER II

REQUIREMENTS DEFINITION

2.1 Introduction

This chapter establishes the requirements for the development of a personal computer (PC) Computer-Aided Design (CAD) package to assist a user in solving complicated control system design and analysis problems. After the System Planning Phase the Requirements Definition and Analysis Phase is the second important step in the System Development Life Cycle (14). A thorough requirements analysis is necessary prior to any large scale software development. This analysis insures that all issues of concern have been thoroughly addressed and evaluated prior to the Design Phase. It is important to note however, the entire design process is iterative in nature. A limitation or capability discovered during the Design/Implementation Phase may cause a basic requirement to be changed. Equally, a nuance discovered during the Testing Phase may well cause changes to the Design/Implementation and, in turn, the requirements Definition. As a result the requirements may change during the life cycle. This chapter is organized into six major sections:

- design environment
- functional requirements
- interface requirements
- software engineering requirements
- testing requirements
- 'other' requirements

In order to accomplish a requirements analysis the user environment must be evaluated. This evaluation is accomplished in Section 2. The functional requirements of ICECAP-PC are specified in Section 3. The functional requirements are divided into priority one and priority two

tasks. The key element of the user/software system interface is discussed in Section 4. The fifth section discusses the various methods of software development. Section 6 contains the requirements for testing the software system. And finally system portability is discussed.

2.2 Design Environment

The primary environmental consideration is the microcomputer hardware and software capabilities. Because of the limited size of the processor, the size of the software tool that may be implemented, the amount of information available to the user, and the speed of the software is limited.

The limitations may be grouped into three major areas: memory addressing, storage capabilities, and the capabilities of the operating system (17). The software system shall be defined, designed, implemented, and tested with these limitations in mind. These limitations are discussed in detailed in the Preliminary Design, Chapter III.

2.3 Functional Requirements

The functional requirements of ICECAP-PC focus on the subroutine development, which includes the design and the implementation of the modules that actually accomplish the numerical analysis. These modules will also provide on-line assistance and error protection/detection. The relationship between the subroutines, the user, and the user interface is illustrated in Figure 2.1 The interaction between the subroutines and the user interface is detailed in Figure 2.2

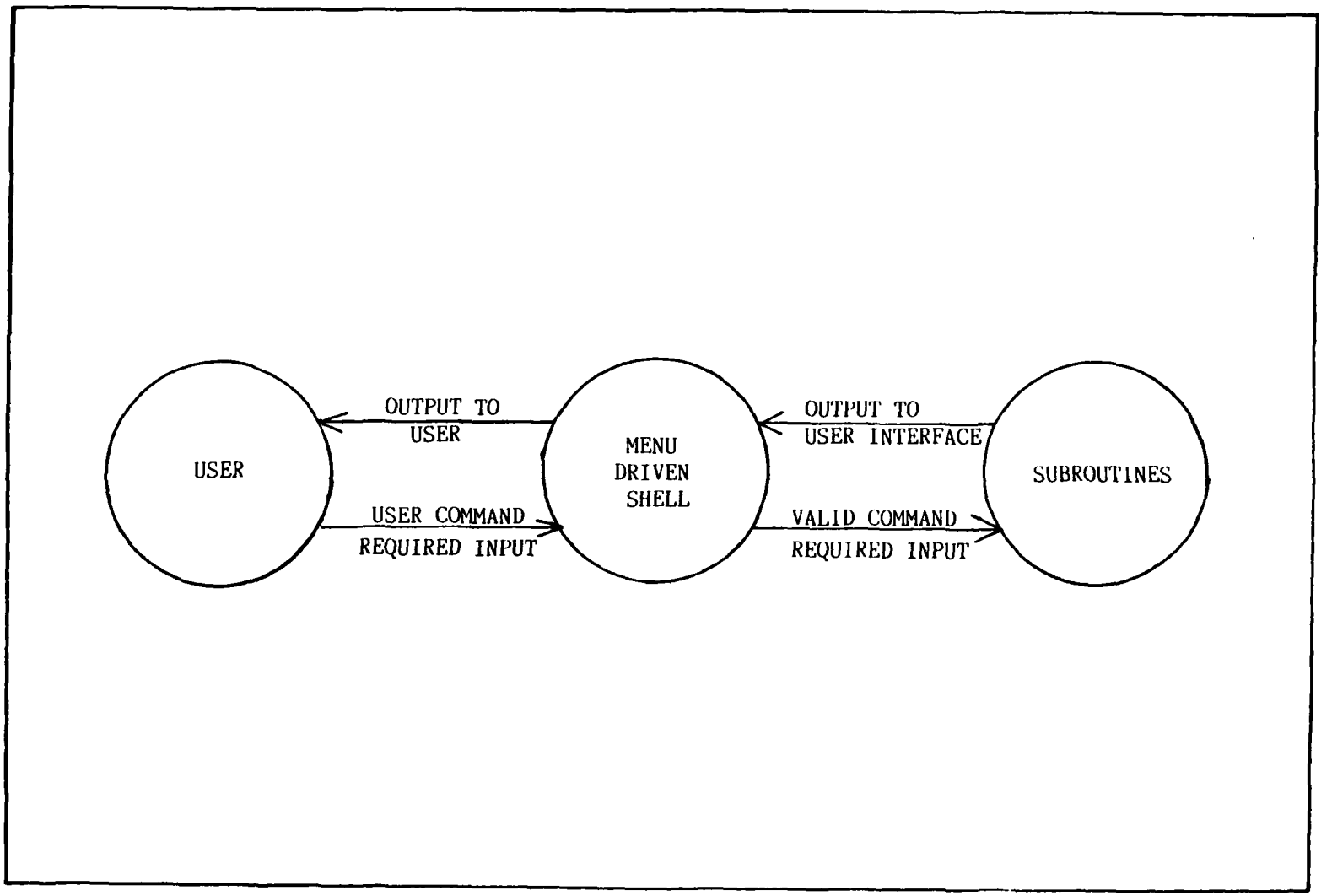


FIGURE 2.1: Relationship Between User, User Interface, and Subroutines

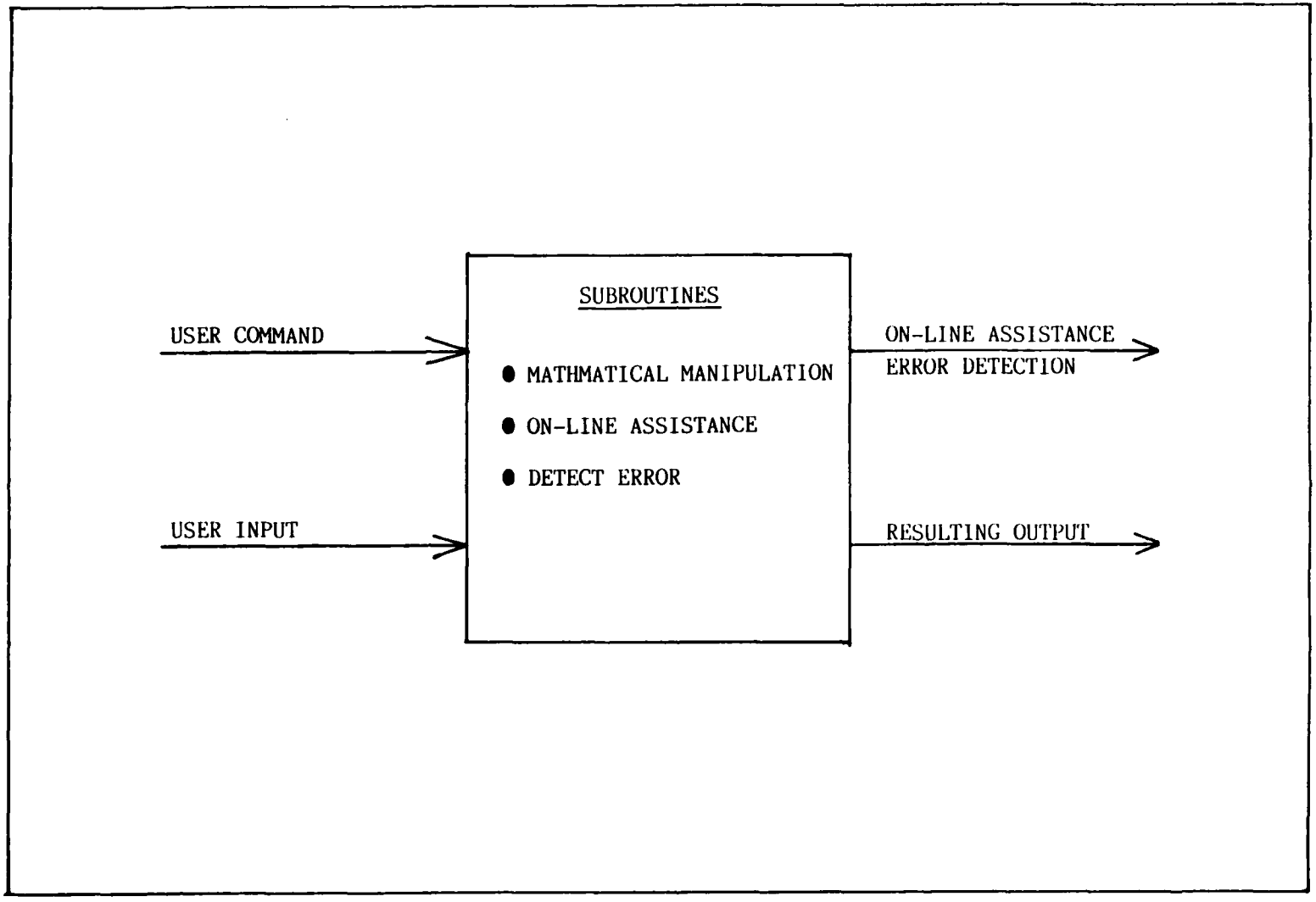


FIGURE 2.2: Subroutines

The following two sections indicate the priorities established for the development of the subroutine software. Parisi's MS Thesis also contains a list of those requirements (20). Due to the time limitations of this effort, the modules are divided into two different groups, priority one and priority two. The priority one modules are those considered necessary for system operation and basic control system design and analysis in the continuous-time domain. These modules are an appropriate tool set for a basic level feedback control course (8). All of the transfer function manipulation modules, polynomial manipulation modules, and matrix manipulation modules are considered priority one. They are considered priority one because they represent basic algorithms applicable to many different diciplines. The time response analysis modules are priority one because they provide a basic measurement of system performance.

The priority two modules consist of the remainder of the continuous-time domain modules and the basics of the discrete-time analysis tools. These modules are the logical choice for follow-on development.

In order to support the follow-on development and maintenance of this software product, the overall structure of ICECAP-PC shall support the addition of new modules. Due to the ultimate size of ICECAP-PC, a module must be thoroughly tested prior to integration with the main program. Module level testing will test the modules ability to accomplish the desired function, as well as test for compatibility with the main program.

2.3.1 Priority One Modules. The following are the priority one requirements. These modules are necessary for initial program operation

and basic control system design and analysis work (8). Where applicable, references are given for possible sources for the necessary algorithms.

Data input modules:

- Polynomial numerator and denominator input
  - -- Factored form
  - -- Polynomial form
- Enter the following transfer functions in either a factored form or a polynomial form
  - -- Forward transfer function (GTF)
  - -- Feedback transfer function (HTF)
  - -- Open loop transfer function (OLTF)
  - -- Closed loop transfer function (CLTF)
- Matrix input

Manipulation of input:

- Manipulate transfer functions
  - -- Form OLTF from GTF and HTF (5, 10, 13)
  - -- Form CLTF from OLTF (5, 10, 13)
  - -- Form CLTF from GTF and HTF (5, 10, 13)
- Polynomial manipulation (10)
  - -- Polynomial addition
  - -- Polynomial subtraction
  - -- Polynomial multiplication
- Matrix manipulation (10, 18)
  - -- Matrix addition
  - -- Matrix subtraction
  - -- Matrix multiplication

-- Matrix transposition

-- Matrix inversion

Time response analysis:

- Accept choice of forcing function step, ramp, impulse (5, 13)
- Calculate F(t) versus time (5, 13)
  - -- Tabular listing of F(t) versus t
  - -- Plot the output of F(t) versus t on the terminal.
  - -- Plot the output of F(t) versus t on a printer.
- Calculate Figures of Merit (5, 13)
  - -- Rise time (Tr)
  - -- Peak time (Tp)
  - -- Peak Magnitude (Mp)
  - -- Settling time (Ts)

Frequency response analysis:

- Calculate the magnitude versus frequency (5,13)
  - -- Log scale
  - -- Linear scale
- Calculate the phase versus frequency (5, 13)
  - -- Log scale
  - -- Linear scale
- Tabular output of frequency response
- Plot the output on the terminal.
- Plot the output on a printer.

Root-Locus analysis:

- General Root-Locus (5, 10, 13, 20)
  - -- Plot the root-locus on the terminal.
  - -- Plot the root-locus on a printer.
  - -- Tabular output of root-locus
- Find the roots of interest (5, 10, 13, 20)
- Find the gain of interest given zeta (5, 10, 13)
- Find zeta of interest (5, 10, 13)

2.3.2 Priority Two Modules. The following are the priority two requirements. These modules shall enhance the utility of the priority one modules (20). Where applicable, references are given for possible sources for the necessary algorithms.

Laplace domain (5, 13):

- Laplace transform
- Inverse Laplace transform

Partial fraction expansion (5,13)

Discrete Control Analysis (1, 5, 13):

- Z domain
  - -- Z transform
  - -- Inverse Z transform
- Root-locus analysis. The details of discrete control root-locus analysis are the same as the continuous case.
- Frequency response analysis. The details of discrete control frequency response analysis are the same as the continuous case.

- Time response analysis. The details of discrete control time response analysis are the same as the continuous case.

Modern Control Theory Algorithms (5, 18)

Block Diagram Manipulation (5)

2.4 Interface Requirements

The user is assumed to be a control system engineer or engineering student who is acquainted with the host computer, at a basic level. Additionally, it is assumed that the user is familiar with the terminology associated with computer operating systems and control system design and analysis.

The interface between the user and the software system "is often the most important aspect which determines the success of a software package" (20). It is through this interface that the user enters the data and selects the required operation. Also the results of the required operation are provided to the user through this interface. The most successful user interface requires an interactive input/output with the software system, along with carefully designed error protection and detection (20).

2.4.1 Interactive Input/Output. The software system shall communicate interactively with the user. The user inputs data and selects the operation and the software system in turn provides the user with the results. The communication between ICECAP-PC and the user should be "clear" and "concise" while at the same time be user-friendly. If at any time the user runs into confusion about what the program is asking for, or what tools are available, there should be on-line

documentation accessible through the HELP subsystem. This assistance should not interfere with the user whose skill level is such that this type of assistance is a hinderance. The software system shall complement the knowledge of the user and the user shall not have to re-learn control theory in order to use this system.

2.4.2 Error Protection and Detection. There are two different types of errors that may be encountered, input error or invalid command. When the user enters incorrect data, ICECAP-PC shall notify the user of the error. The error message shall tell the user what type of error has been made and provide a "polite" reminder of the correct form.

The command interpreter should only allow valid commands to begin execution. If the user were to enter a command that is not valid, the system should provide a means to exit that command gracefully, as well as provide a message to the user that the selected command is not valid. The results to the point of the invalid command should not be affected by the exit operation. If an exit to the operating system is required, the data in the system to that point should not be lost. The exit to the operating system may either be intentional or unintentional. In any case, there should be no way that an incorrect entry by the user could cause a massive failure of the program or cause damage to the system.

2.5 Software Engineering Requirements

ICECAP-PC shall be designed with sound software engineering principles. Where software engineering is defined as "the technological and managerial dicipline concerned with the systematic production and maintanance of software products" (7). This shall eliminate the problem

of little or no documentation that is encountered in many software systems. ICECAP-PC documentation shall be developed concurrently with the software itself.

ICECAP-PC shall be modular. ICECAP-PC shall be divided into sub-tasks that are easier to debug and test. The modules shall be general, this allows the modules to be used in several different locations in the software system organization. Operating system and hardware unique code shall be localized for ease of modification. When new modules are added to existing software these modules shall not interfere with the operation nor the accuracy of the existing system.

ICECAP-PC shall be fully documented. The complete documentation package shall include a data dictionary, structure charts, and code comments (15). Because this is a follow on effort the documentation methods of the previous work shall be utilized.

The data dictionary shall provide a central repository for data about the system software. ICECAP-PC data dictionary shall be updated as more definition is added to the software system.

The documentation package shall contain structure charts that reflect how the different ICECAP-PC modules are interconnected. These charts will also show how the parameters are passed between modules.

ICECAP-PC shall be self-documenting. The program listing shall contain sufficent comments to allow follow on efforts to understand the abbreviations and the logic of the previous efforts.

2.6 Testing Requirements

ICECAP-PC shall be tested throughout its development. The testing process shall be performed concurrently with the design, development,

and the debugging phases of this effort. The functional requirements of ICECAP-PC shall be tested using known test cases. It shall be determined whether or not the results of ICECAP-PC are consistant with either TOTAL or ICECAP.

ICECAP-PC results shall be considered consistant with those produced by either TOTAL and ICECAP when the differences between the results can be attributed to truncation or round-off error. Both truncation and roundoff error are inherent to computers of different wordlength. TOTAL is hosted on the ASD/CYBER which is a 60-bit wordlength machine, ICECAP is hosted on a VAX 11/780 which is a 32-bit wordlength machine (20). Microcomputers are either 8-bit or 16-bit wordlength machines. The personal computers used in this design effort, contain a 16-bit microprocessor.

The user/software interface requirements shall be tested by the authors and potential users of ICECAP-PC, graduate level controls students. The users shall evaluate the interactive input/output and the error protection/detection features of ICECAP-PC.

2.7 Additional Requirements

Portability. One of the goals of this development is to provide a control system design and analysis tool that is available to a large number of users on several different types of microcomputers. As a result, portability to other operating systems and hardware environments is required. Portability is determined by a tools dependence on the language it is written in, the operating system it works with, and the hardware it operates on. The portability of a tool can be greatly enhanced by choosing a popular higher order language, operating system, and hardware environment. Additionally, the portability of the tool can

be increased by minimizing the various dependencies. If these dependencies are localized, the tool may be 'easily' modified for use on other operating systems and hardware environments.

2.8 Summary

This chapter analyzes and establishes the requirements for a CAD control system design and analysis tool. The top-level functional requirements are provided for the software system development. The fundamental requirements that the ICECAP communicate interactively with the user, provide error detection, and error protection were established. The system development requirements which specified a structured environment were also established. Finally the testing requirements for the software system were specified. The requirements specified in this chapter provide the foundation for the entire design effort and supports the framework for any follow on effort. In Chapter III the decisions made during the preliminary design phase of this effort are described. These decisions focus primarily on resource selection and the user interface.

CHAPTER III

PRELIMINARY DESIGN

3.1 Introduction

The purpose of the Preliminary Design Phase of ICECAP-PC is to select the resources necessary for this software development and to organize the characteristics and the requirements described in the previous chapter into a functional structure. The goal of this chapter is to produce a Preliminary Design Document, which is composed of this chapter, the system structure charts, and the system data dictionary. The system structure charts are in Appendix C and the system data dictionary is in Appendix D. This chapter is divided into two major sections: resource selection and system design.

The resource selection portion of this chapter deals with the selection of the hardware and the software required for the fourth phase of the software life cycle, Implementation (14). The system design section focuses on the user interface. The user interface includes on-line help and information transfer.

3.2 Resource Selection

This section discusses the selection of the system resources, namely the operating system, the design language, and the system hardware.

3.2.1 Operating System. The operating system is the interface between the software and the computer hardware. Ideally, the operating system should not be a major design consideration. The higher order language used in system design should be standardized such that the

operating system is transparent to the language used. In reality, the operating system is not transparent and must be considered in the system design. The selection of the operating system is critical because it significantly affects the portability of the software package. The more popular the operating system the more portable the software package.

The software package MICROSDW was written under the Digital Research CP/M operating system (17). MICROSDW is a software development environment that facilitates the creation of a user interface. This user interface will be used as the umbrella or executive for ICECAP-PC. In order to increase the portability of ICECAP-PC the operating system to be used in this design effort will be MS-DOS.

In a recent report by Datapro the MS-DOS operating system "dominates, and its popularity is growing" (28). By implementing ICECAP-PC in MS-DOS the portability will be maximized. Based on the work done by Moore with MICROSDW, ICECAP-PC will be able to operate with the CP/M operating system (17). This effort will expand the portability by providing the capability to operate with MS-DOS.

3.2.2 Design Language. The programming language for ICECAP-PC was chosen based on the following criteria:

- the design language must be available for multiple operating systems.
- the design language must be available for multiple microprocessors.
- the compiler/linker speed.
- executable code speed and accuracy.
- numeric accuracy.
- memory management support.

- ease of use.
- popularity. (20)

If the selected design language is available for multiple operating systems and multiple microprocessors, ICECAP-PC will be portable to other systems. When the design language is implemented in more than one operating system however, care must be taken to insure that the language constructs and syntax are the same under the different operating system implementations. If they are not the same, the portability will be limited.

The speed and efficency of a higher order language can be a significant factor in the performance of that language. A slow higher order language will hamper the further development of the CAD package. There are two major factors that contribute to the speed and efficency of the language, code type and the inherent efficency of the language. There are two basic categories of code, interpretive and compiler. Interpretive language implementations must first process through a preliminary evaluation sequence prior to the actual calculation or evaluation. The compiler implementation on the other hand accomplishes the preliminary evaluation during the compilation. Inherently, the compiler type language is faster than the interpretive type language. There is a great deal of variation in speed within the categories themselves. Two different compiler type languages PASCAL MT+ and TURBO-Pascal vary greatly in speed. Using similar compiler options, Pascal MT+ requires twenty-three minutes to compile 2800 lines of code, while TURBO-PASCAL only requires about two minutes and thirty seconds to link and compile the same source files (with minor modifications for compatibility with TURBO-Pascal) (17).

Due to the limited memory available on most of the microcomputers in use today, the speed and the size of the executable programs created by the higher order language is very important. The smaller and more compact the code, the larger the program can be before various memory management techniques are employed. These techniques include chaining, overlays, and the execution of external programs (20). The program used in the above illustration required 42% less memory in the TURBO-Pascal implementation than in the PASCAL MT+ implementation (17).

Numeric accuracy is an important factor to consider in the selection of a design language for ICECAP-PC because of the numeric nature of the control system design and analysis problems. Ideally, the greater the number of digits of accuracy in floating point numbers provided by the language the better. But realistically there is a trade-off between processing speed, memory, and numerical accuracy. If the language supports hardware floating point calculations and the host computer contains the necessary floating point hardware, the processing time can be cut significantly (17, 25).

Memory management support is of particular concern for this software development due to the large size of the application software that will eventually be a part of ICECAP-PC. Memory management techniques include: chaining, overlays, and the execution of external programs. Chaining loads a new program containing only program code with no language libraries from disk into the memory space formerly occupied by the previously executed code.

Overlays allow one to create programs much larger than can be accomodated by the computer's memory. A number of subprograms are placed into a file that is separate from the main program file. These separate

subprograms are then loaded automatically one at a time into the same space in memory. Execution of an external program loads a new program into memory without preserving the previous program. Caution must be used when using memory management techniques because they slow the program execution while at the same time providing the benifit of increasing the size of the program that may be executed (25).

The selected design language should be 'easy to use'. If an editor is provided it should be easy to use or similar to a standard wordprocessing editor. The language should be user friendly. User-friendliness is measured subjectively by the individual (27), but as a minimum the language should be able to provide help to the user. More importantly however, the compiler should be able to state the nature of an error and pinpoint the location of the error in the code. This is particularly important for the development of long programs such as ICECAP-PC.

The language selected for ICECAP-PC must be a popular language that is readily available to all of the potential users of ICECAP-PC. ICECAP-PC is to be public domain software and as such should be open to improvements and modification. In order to facilitate this, the language should be relatively inexpensive, and mature enough so that the 'bugs' in the language are reduced and continued customer support and availability is assured.

The program CONTROL-CAD was developed with PASCAL MT+. PASCAL MT+ was selected over thirty eight other design languages because it provided the necessary requirements of structure, speed, memory management and portability (20). At that time TURBO-Pascal was not available for evaluation. MICROSDW was developed with TURBO-Pascal.

TURBO-Pascal was selected for the MICROSDW development for the following reasons, Table 2.1

Table 2.1 (17)

Comparison of Pascal MT+ and TURBO-Pascal

	TURBO-PASCAL	PASCAL MT+
Operating System	CP/M, CP/M86 MS-DOS, PC-DOS	CP/M
Compiler/Linker Speed	2:30 for 2800 lines	23:00 for 2800 lines
Executable Code Size	22K (for above)	38K
Floating Point Numeric Accuracy	11 digits	6.5 digits
Memory Management	overlays, chaining external program execution	overlays

TURBO-Pascal was chosen over other programming languages for the development of ICECAP-PC for the same reasons it was chosen for MICROSDW. Borland International has made several improvements to their program since Table 2.1 was developed, the most significant for the ICECAP-PC application is that TURBO-Pascal now supports the 8087 math coprocessor. This increases the floating point numeric accuracy from 11 digits to 16 digits, and "significantly" increases the speed (25).

3.2.3 System Hardware. The development and subsequent operation of this program requires a computer that is capable of using the MS-DOS, version 1.0 or greater, or the PC-DOS, version 1.0 or greater. This computer must have at least 64K RAM and a disk drive, both of which are required to run TURBO-Pascal. Improved performance would be noted if a hard disk or a disk emulator were used instead. Various files and

records will be accessed throughout the operation of ICECAP-PC, as a result, a tape storage device would not provide the desired performance.

There is an additional requirement of an 80 column by 24 line monochrome monitor. This monitor will facilitate the transfer of information to the user. The program ICECAP-PC will be implemented primarily for a monochrome monitor, however, the program is capable of supporting high resolution graphics. The monochrome monitor was chosen for the initial implementation of ICECAP-PC because most computers have a monochrome monitor. The systems with high resolution graphics monitors are capable of displaying the monochrome characters, while monochrome monitors are not capable of correctly displaying the high resolution graphics characters.

3.3 System Design

This section contains the functional structure that will be used as the foundation for the detailed/implemented design in Chapter IV.

Because the design process is iterative in nature, this is a 'living' document. It reflects the most recent ICECAP-PC design.

3.3.1 Functional Requirements.

The requirements of the User Interface and the Installation programs for ICECAP-PC are unchanged from the MICROSDW effort. The Structured Analysis and Design Technique (SADT) contained in reference 17 apply to this effort.

3.3.2 User Interface Design.

Background. Perhaps the most important aspect of the design problem is that of the user interface. For the development of this important aspect of the program an existing piece of software (17) shall be

used. This software has been and will be refered to as MICROSDW. The objective of MICROSDW was to perform the initial development and implementation of a microcomputer software development environment. MICROSDW implements a menu (command/key word) driven user interface that may be adapted for use in ICECAP-PC. MICROSDW is essentially two different programs, an installation program, BUILDDAT, and a user interface program, MICROSDW. Both of these programs were derived from a program called CONTROL-CAD (20).

Current Implementation. The installation program, BUILDDAT builds the initialization files for MICROSDW from the ASCII text files. These initialization files, HELP.SYS and MICROSDW.SYS, describe the hardware characteristics and the menu structure. Once MICROSDW is compiled and run, the program prompts the user for inputs by displaying the valid keyword options. Once the keyword is entered, the program checks for the validity of the keyword, and executes the functions selected by the user. In this implementation of the keyword/commandword program execution, the program prompts the user for input and allows selected abbreviations to be used in the place of the full keyword/command word. These abbreviations are highlighted on the screen for the user.

On-line Help. The ICECAP-PC is designed to be user-friendly. A very important aspect of user-friendliness is the availability of on-line help. Help is provided in three ways:

- The system prompts the user for his choice.
- The system provides error analysis and correction. ICECAP-PC will detect an operator error and offer assistance for correction.
- The system will provide user requested assistance on a specific topic.

The user who is unfamiliar with the system can request help on the system in general or on a specific topic. ICECAP-PC will present the user with a menu of available choices and prompts for input. When the user makes an error inputting data, ICECAP-PC determines the error and provides a prompt for correct input. An experienced user will not request system or topical help but will still be prompted for input and error analysis will still be provided.

Information Transfer. Standard interaction with MICROSDW presents the results of desired operations to the terminal. Output to the printer or to the temporary files are the other possible destinations. The software should be written to take advantage of these other destinations. The interface to different types of printers needs to be constructed, and procedures for routing the output needs to be developed.

Improvements. The MICROSDW software environment provides an excellent basis for the development of a computer aided design tool for control system design and analysis. However before MICROSDW may be used in this fashion certain changes and improvements must be made to the program.

The terminal parameter files must be developed for systems other than the Rainbow 100. The previous effort did not include the parameters for any other systems.

The MICROSDW program must be modified to work with the MS-DOS operating system. The previous effort was only designed to work with the CP/M operating system, additionally MICROSDW was only tested with CP/M.

3.4 Summary

In order to maximize the portability of ICECAP-PC, MS-DOS was chosen as the operating system and the reasonably priced TURBO-Pascal was chosen as the design language. The hardware that will be used during the design phase of this effort are the Z-100 dual floppy disk system, Z-100 hard disk system, IBM-PC, IBM-AT, and the IBM-XT. In addition to the selection of the resources for this design effort, this chapter discussed the system design aspect of the user interface. Chapter IV will discuss the structure and implemented design of ICECAP-PC subroutines, as well as the interaction of the those subroutines with MICROSDW.

CHAPTER IV

IMPLEMENTED DESIGN

4.1 Introduction

The preceeding chapters have defined the needs and the specific goals for this software development. This chapter discusses the decisions made during the Implementation Phase of the ICECAP-PC software development. The purpose of the Implementation Phase is to translate the Detailed Design into the language selected for implementing the software, in this instance TURBO-Pascal. The design documentation is the primary input to the Implementation Phase. In this case, the design documentation consists of the Structure Charts, Appendix C, and the Data Dictionary, Appendix D. The output of the Implementation Phase is the actual source code, Appendix E.

It is important to note that the entire design process is iterative in nature, a particular nuance discovered in the Implementation Phase may well cause the basic design to be fundamentally changed or at least modified. Once the implementation is considered complete it is then tested, these tests may cause changes to the implementation and in turn the basic design. As a result, the Design, the Implementation, and the Testing Phases of a program are inseparable and must be considered as a whole.

Specifically, this chapter discusses the modifications made to MICROSDW, the translation of the Detailed Design into TURBO-Pascal, the design approach, the user interface design, and the major functional areas of ICECAP-PC in detail.

4.2 Modifications to MICROSDW

The primary modifications that were made to MICROSDW focused on the operating system and the computer system unique software. The software was modified to work on the resources selected in the previous chapter, specifically, the Z-100 and the IBM-PC hardware Systems which use the MS-DOS and the PC-DOS operating systems respectively.

4.3 Translation of Detailed Design to TURBO

During the Design/Implementation Phase of this development it was discovered that the version of TURBO-Pascal for the IBM-PC/AT/XT is different from the versions for any other hardware system. This difference affects the control of the terminal and is not documented in the TURBO-Pascal Reference Manual (25) or in open literature. The code modifications required to control the terminal for the IBM-PC/AT/XT may be found in Appendix E of this document.

Because of the differences in TURBO-Pascal, two separate versions of ICECAP-PC were developed, one version for the IBM-PC/AT/XT and the second for all other systems.

4.4 Design Approach

The design approach taken in this effort was a 'top-down' design approach. This means the program was viewed from the highest possible level and then was broken down consecutively into finer and finer levels of detail. If this top-down design approach had not been taken the modular approach requirement laid out in Chapter II would have been very difficult if not impossible to meet. The finest level of detail in this approach is the level that contains the general modules which are used in several different areas of the design.

An example of this approach is, at the onset of this effort it was not known that the partial fraction expansion procedures would be required as a priority one module, Chapter II. It was only when the time response option was studied in detail that this requirement was discovered.

The program was initially divided into twelve functional areas:

- Exit from ICECAP-PC to the operating system.
- Provide on-line help to the user.
- Define a transfer function, polynomial, or matrix.
- Display various functions or results on the terminal.
- Copy transfer functions, polynomial, or matrix from one location to another.
- Modify a polynomial, transfer function or matrix without redefining the entire object.
- Be able to form the OLTF or the CLTF from the GTF and the HTF.
- Store the ICECAP-PC files to user specified files.
- Recover ICECAP-PC files from user specified files.
- Toggle the ICECAP-PC boolean switches from the terminal.
- Print various functions or results on a printer.
- Change the analysis plane or the sampling time.

Each of these areas was then further divided into lower level functions. Next, the necessary algorithms and procedures to perform each of the functions were either located or developed. Finally, the specific procedures were written. It was realized at the onset of this effort that all twelve of the functional areas might not reach the coding stage, because of this the areas were prioritized. In general, if ICECAP-PC could

operate without one of these areas it was placed at the bottom of the list, equally if ICECAP-PC could not operate without the area it was placed at the top of the list. The priority is reflected by the order which the areas are presented above. Areas one thru nine were fully designed and implemented while the last three areas did not progress beyond the design stage. The relationship between ICECAP-PC and twelve functional areas is shown in Figure 4.1, each of the areas described in detail in Section 4.6. Before any of the major functional areas were designed and implemented the human interface for ICECAP-PC was designed and coded.

4.5 User Interface Design

ICECAP-PC is designed to communicate interactively with the user. The user inputs the data and selects the operation and ICECAP-PC in turn, provides the user with the results. If at any point in time the user runs into confusion about what ICECAP-PC is asking for, or what tools are available, or what those tools do, on-line assistance is available at all levels. This assistance is only presented to the user at his request, as a result the experienced user is not hindered. Each of the twelve functional areas has on-line assistance available.

The second way that ICECAP-PC is designed to aid the user is with user input prompts. Anytime information is required from the user ICECAP-PC provides a descriptive prompt. These prompts are provided at all times.

The third type of help that ICECAP-PC is designed to provide the user is also provided at all times, error detection and protection. ICECAP-PC will detect an input error and notify the user of the error.

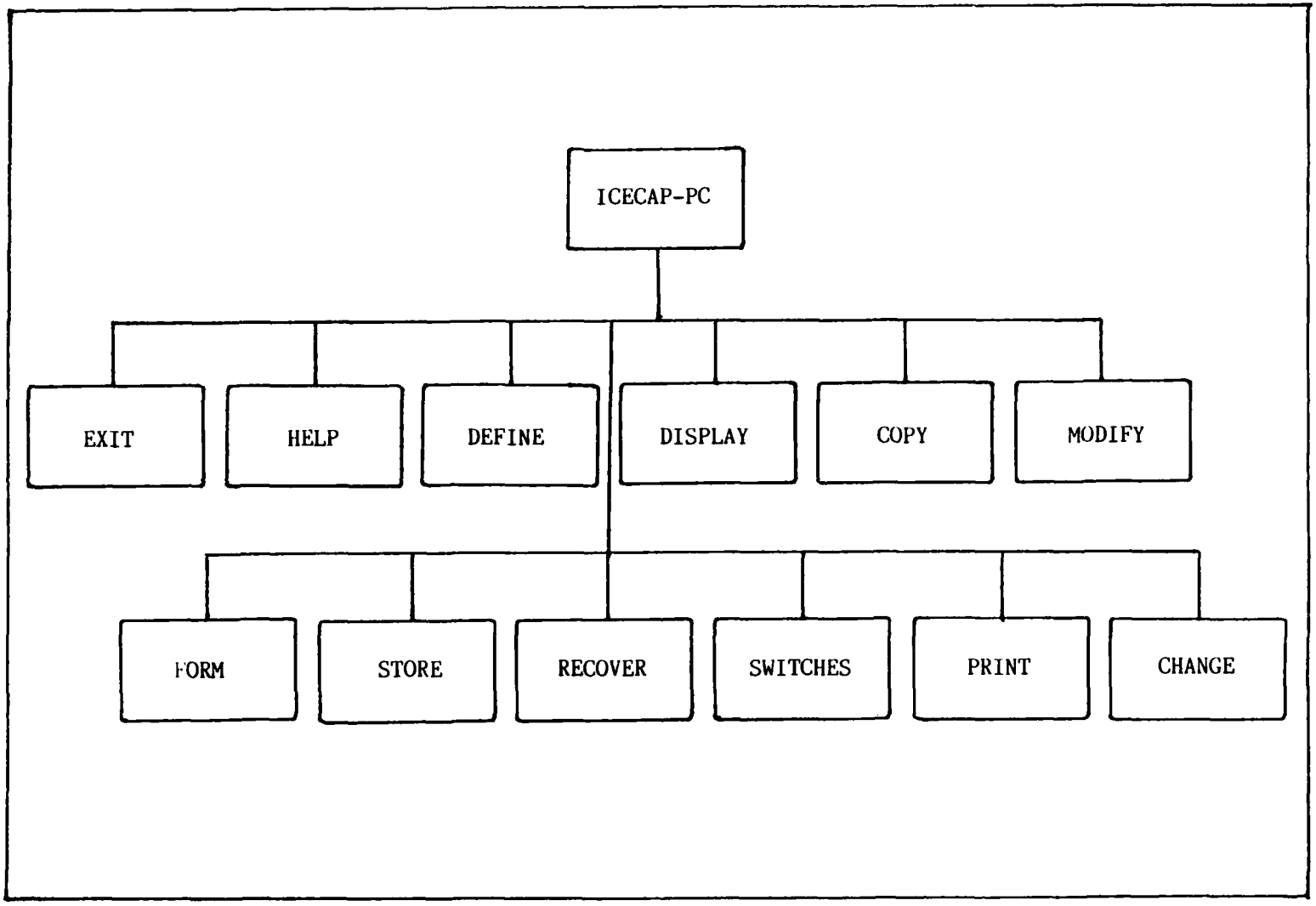


FIGURE 4.1: ICECAP-PC Software System

The displayed error message explains the type of error and offers a reminder of the correct form. ICECAP-PC is also designed to prevent incorrect or invalid commands. If the user command is not valid ICECAP-PC will tell the user this, and prompt the user to re-input the command. In this fashion unintentional exits to the operating system are eliminated.

On-line help and error detection/protection are fundamental to the design of ICECAP-PC and are included in each of the twelve functional areas.

4.6 Major Functional Areas

Each of the twelve major areas is an option in the main menu. The main menu, as it is presented to the user, is shown in Figure 4.2. The main menu is laid out in alphabetical order by column. This was one of several layouts shown to the potential users of ICECAP-PC. The other layouts included, alphabetical order by row, functional groupings of command words, and in the order of the frequency the command would be used. The layout in Figure 4.2 was preferred by more potential users.

Each of the twelve functional areas will be discussed in the order which they are presented in Figure 4.2.

```
CHANGE    DISPLAY   MODIFY    RECOVER   SWITCHES
COPY      FORM      PRINT     STOP      UPDATE
DEFINE    HELP

Enter Option >
```

Figure 4.2: ICECAP-PC Main Menu

4.6.1 Change Command

The change command is used to initialize the change functions of ICECAP-PC. This function allows the user to select the plane of analysis and the sampling time. The lower level menu for the change command is presented in Figure 4.3.

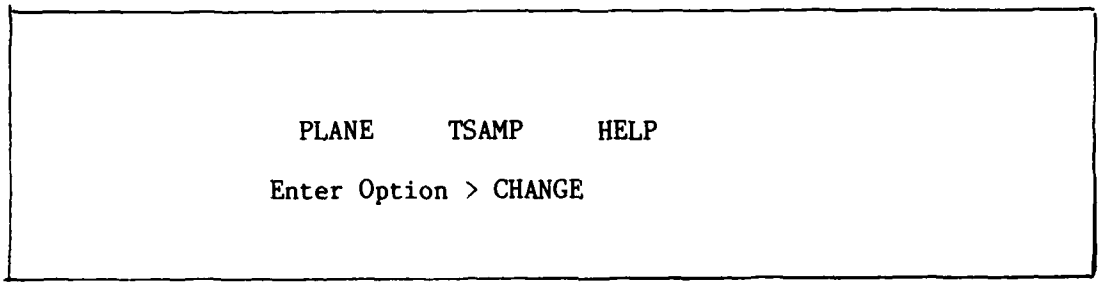


Figure 4.3: Change Command Sub-menu

This option has only reached the design stage and has not been implemented.

4.6.2 Copy Command

The copy function determines the location of the source and destination parameters, reads the information from the source and transfers it to the destination. The copy function does not destroy the contents of the source. After entering the copy option the user is presented with a source menu, Figure 4.4.

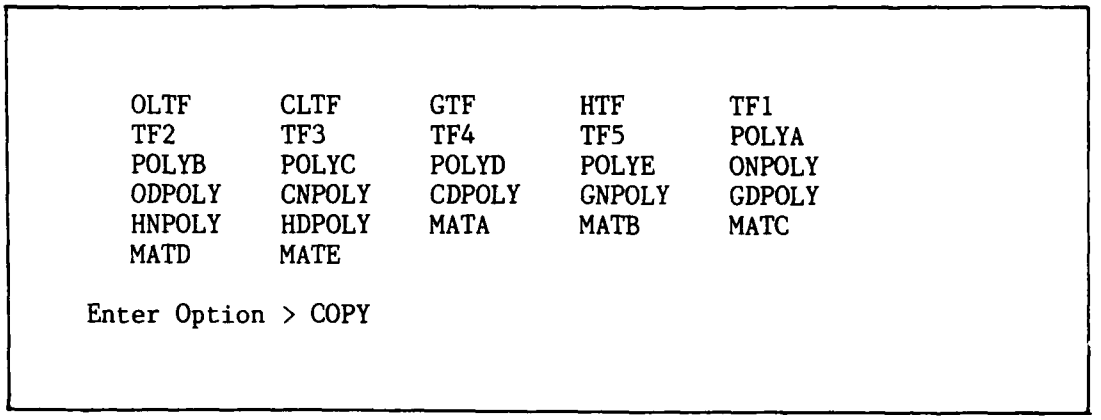


Figure 4.4: Copy Command Source Menu

If the user selects a transfer function as the source he/she is presented with a menu of available destination transfer functions, Figure 4.5. If the user selects a polynomial as the source he/she is presented with a menu of available destination polynomials, Figure 4.6; and if he selects a matrix source he is presented with a menu of destination matrices, Figure 4.7. This prevents the user from copying one type erroneously into a different type. (i.e. A matrix cannot be copied into a transfer function.) The relationship between the various copy commands is illustrated in Figure 4.8.

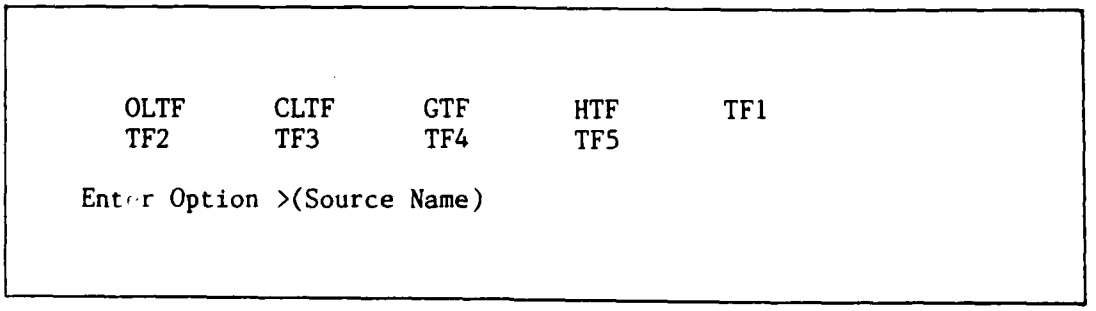


Figure 4.5: Copy Command Destination Menu For Transfer Function

```
POLYA     POLYB     POLYC     POLYD     POLYE
ONPOLY    ODPOLY    CNPOLY    CDPOLY    GNPOLY
GDPOLY    HNPOLY    HDPOLY

Enter Option >(Source Name)
```

Figure 4.6: Copy Command Destination Menu For Polynomials

```
MATA      MATB      MATC      MATD      MATE

Enter Option >(Source Name)
```

Figure 4.7: Copy Command Destination Menu For Matrices

4.6.3 Define Command

The define command routines are designed for the input of transfer functions, polynomials, and matrices. The transfer functions and the polynomials are stored in the disk file 'tf&pols.dat'. The matrices are stored in the disk file 'matrix.dat'. The file structure of both 'tf&pols.dat' and 'matrix.dat' is defined in Appendix D, and illustrated in Figure 4.9. Transfer functions and polynomials may be stored in the same file because a transfer function is merely a numerator polynomial and a denominator polynomial. A transfer function requires two storage locations while a polynomial requires only one. After the user enters the define command he is presented with the menu in Figure 4.10.

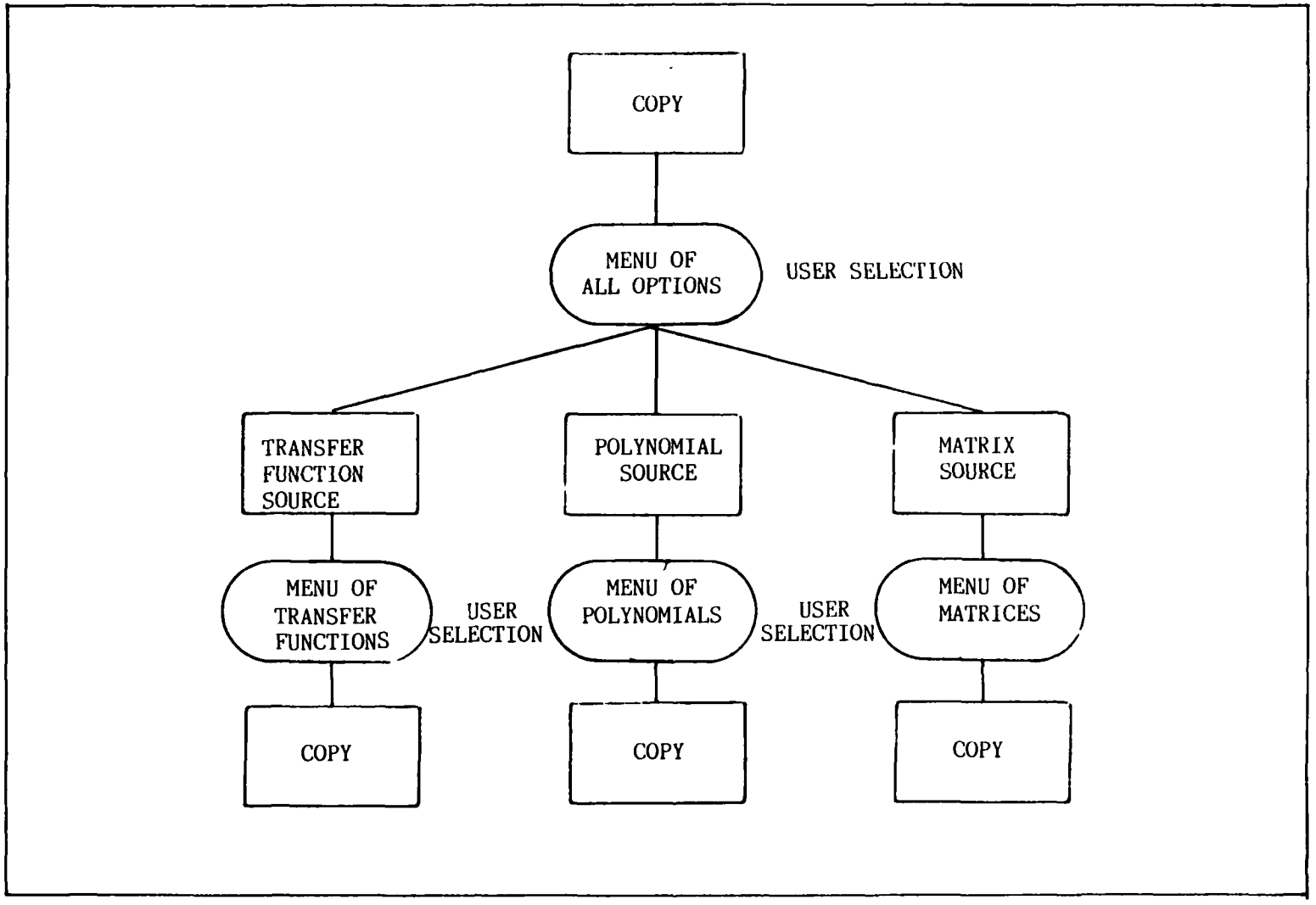


FIGURE 4.8: Copy Command

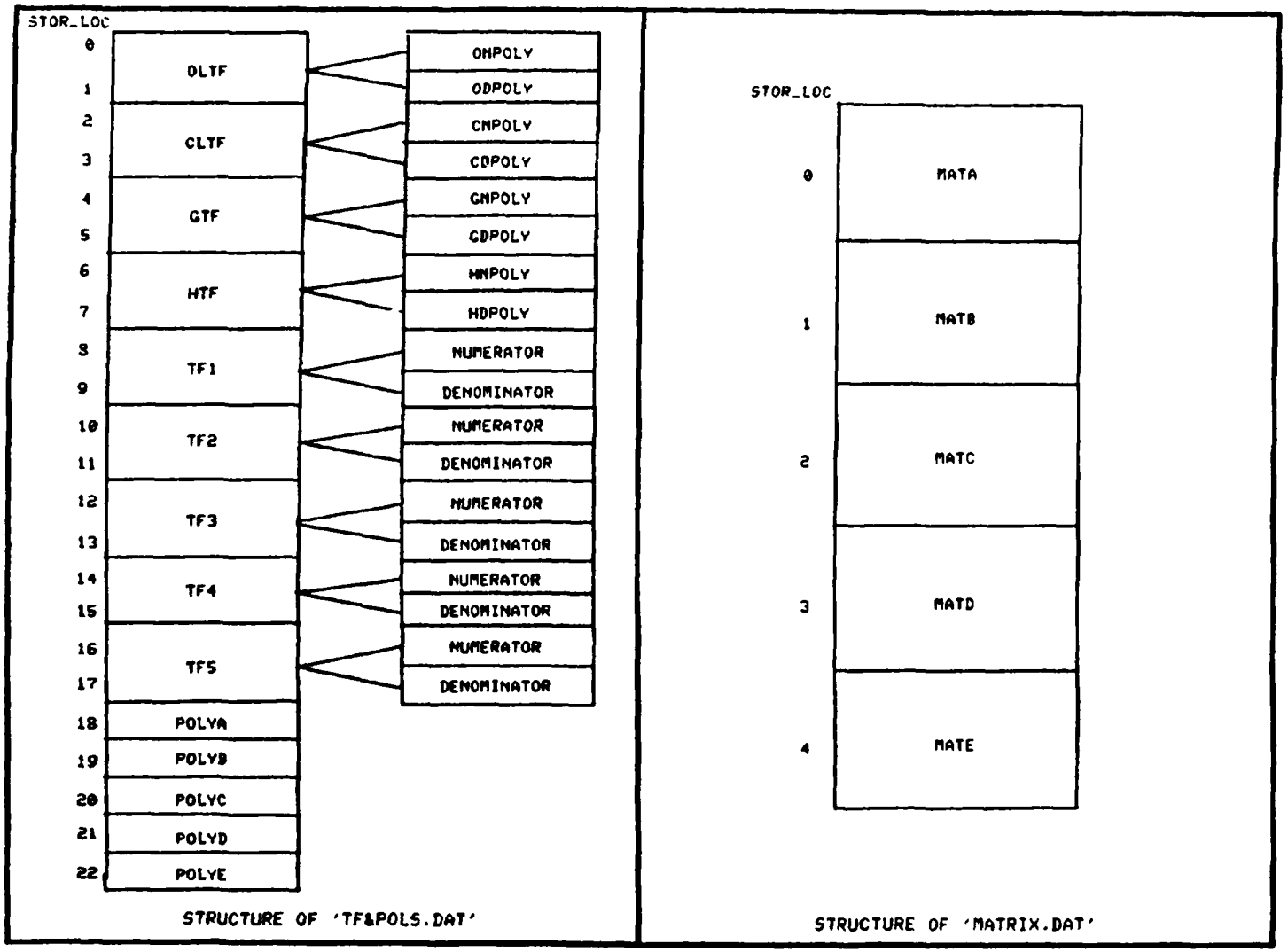


FIGURE 4.9: 'MATRIX.DAT' And 'TF&POLS.DAT' Structure

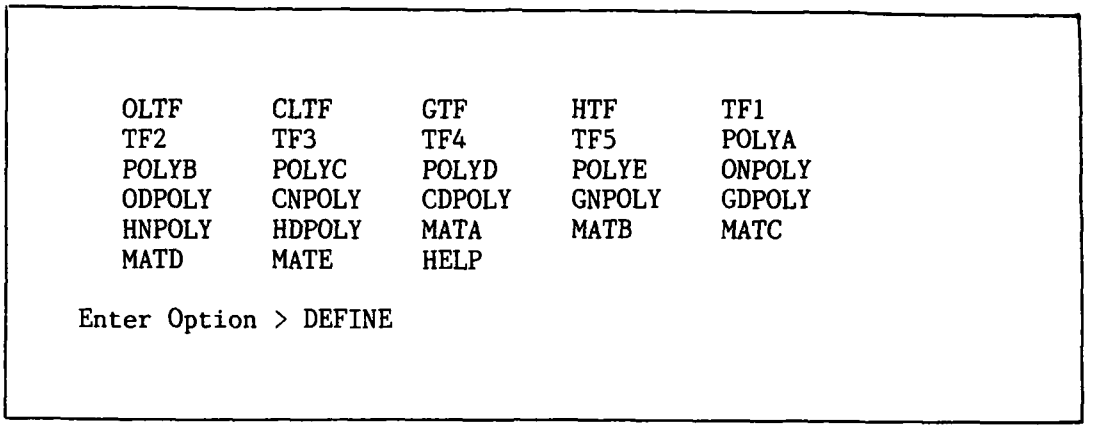


Figure 4.10: Define Command Menu

The user may choose to define a polynomial, a transfer function, or a matrix. A illustration of the entire define command is presented in Figure 4.11. A detailed description of the three different types of definition is presented below.

Polynomial Input

If the user chooses to define a polynomial a submenu will be displayed to the user, Figure 4.12.

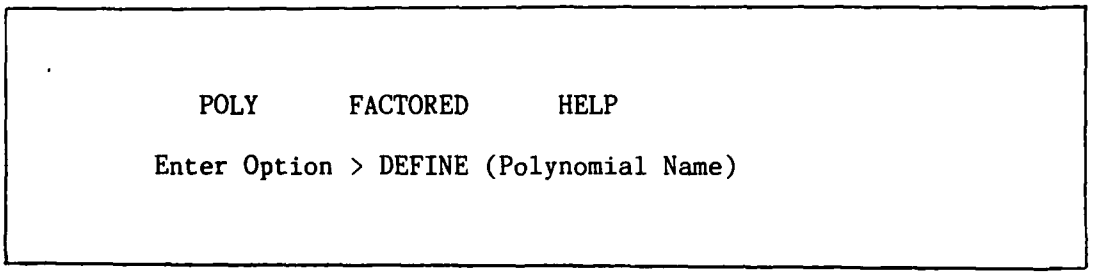


Figure 4.12: Define Command Submenu For Polynomials

If the user selects either poly (input) or factored (input) the polynomial definition routine requests the order of the polynomial from

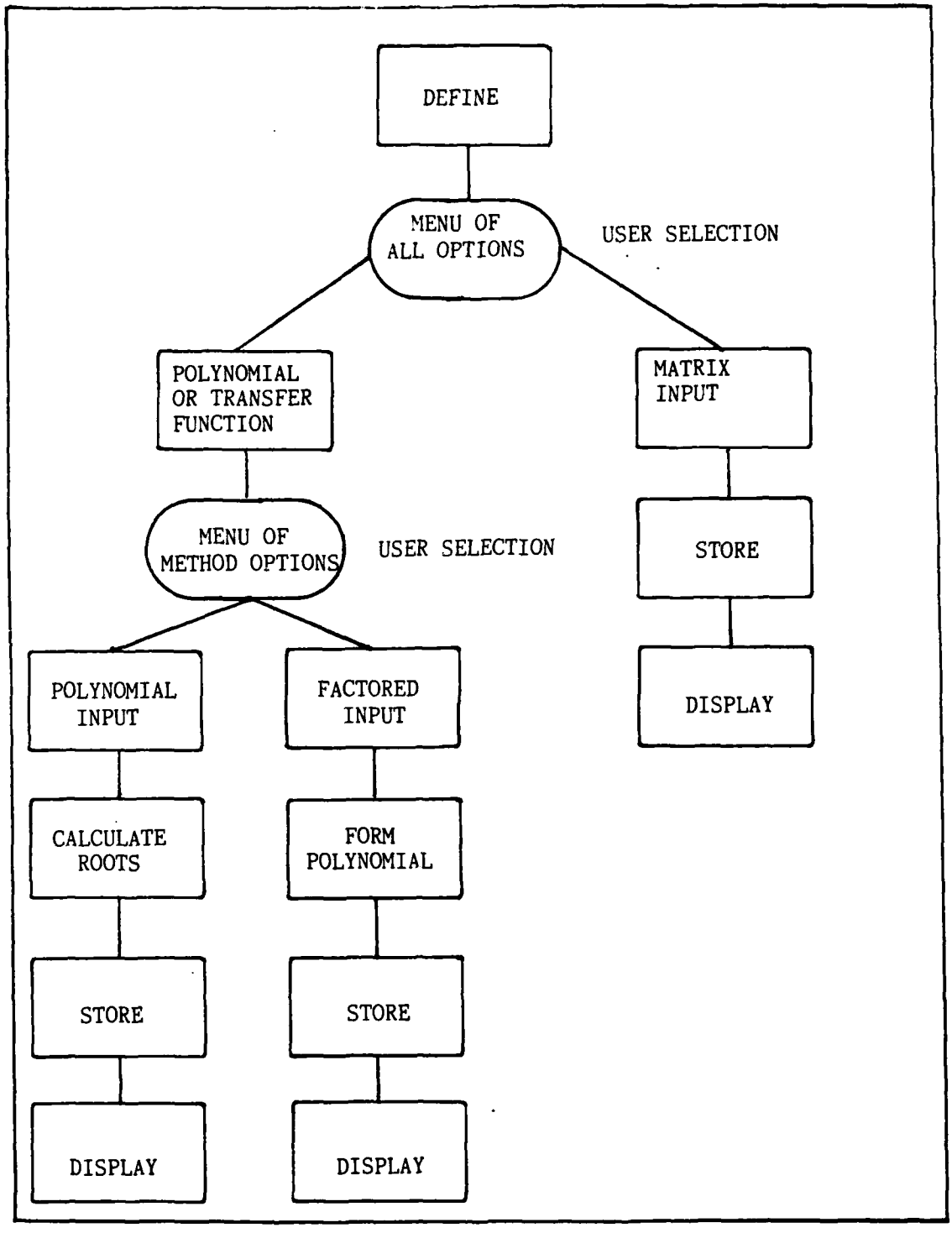


FIGURE 4.11: Define Command

the user. The routine has been designed to accept only integers. If the user's input is an integer, the code then checks to see if it less than the maximum size for polynomials and transfer functions. Once the order of the polynomial has been correctly entered either the poly input routine or the factored input routine is called.

The screen is cleared and a title is displayed indicating which polynomial is being defined. The screen is then divided in half. The left hand side of the screen is formatted for polynomial input and the right hand side is formatted for factored input. Several different displays were considered, the ICECAP-PC display was chosen because it facilitated interactive input/output and was preferred by potential users.

Factored Input

A highlighted area then appears on the screen, this area highlights the information the user provides. The first area to be highlighted is the area to the right of 'constant/gain =', the routine will also provide a prompt at the bottom of the screen. In respose to the prompt the user enters a real number in either exponential format or as a string of numbers. If the input by the user is not a real number the routine will detect the error and display an error message on the screen and ask the user to re-input the number. If the number is valid it is converted to floating point notation and displayed at the proper location on the screen. The highlighted area then moves to the next required user input, the real portion of a root, followed by the imaginary portion. If the user input for the imaginary portion is zero the routine then requests user input for the next real root. If the user input for the imaginary part is anything other than zero the routine

will automatically compute the complex conjugate of the root. If the user enters anything other than zero for the imaginary portion of the last root the routine detects the error. The user is told that there is insufficent room for the complex conjugate and is asked to reenter the number. If at any time during this input process the user desires to exit this routine he should input a dollar sign, $, followed by a carriage return.

After the factored form input is complete the routine will form a polynomial from the roots. The routine will then store both the polynomial and the factored form in the proper location in 'tf&pols.dat'. The defined polynomial is then displayed on the screen from memory, this allows the user to validate the input.

Polynomial Input

If the polynomial input format is selected, a highlighted area also appears on the screen, this area highlights the information that the user is to provide. The first area to be highlighted is the area to the right of 'constant/gain =', the routine will also provide a prompt at the bottom of the screen. In response to the prompt the user enters a real number in either exponential format or as a string of numbers. If the input by the user is not a real number the routine will detect the error and display and error message on the screen and ask the user to re-input the number. If the number is valid it is converted to floating point notation and displayed at the proper loction on the screen. The highlighted area then moves to the coefficient of the first term of the polynomial and prompts the user for input. Once the user enters a valid real number the highlighted area moves to the next reqested data, this

continues until the polynomial is defined. If the user desires to exit this routine he should input a dollar sign, $, followed by a carriage return.

If the leading coefficient of the polynomial is any number other than one, the constant is multiplied by the leading coefficent, all of the other coefficients of the polynomial are divided by the leading coefficient, and the leading coefficient is set to one. As a result, the polynomial that is stored in 'tf&pols.dat' may not look the same as the polynomial entered by the user. The stored polynomial will be the algebraic equivalent of the entered polynomial.

After the polynomial form input is complete the routine will calculate the roots of the polynomial. Several different root finding algorithms were evaluated for use in ICECAP-PC. The first algorithm was Jenkins-Traub, this algorithm is used by IMSL's (11) root finding subroutine which in turn is used by ICECAP. This algorithm is very complex and experience indicated a great amount of time could be spent coding this algorithm with no guarantee of success (16).

The next group of algorithms, which included the popular Newton-Rapson, did not meet a basic requirement of ICECAP-PC. The requirement is, the final root finding algorithm must be able to calculate the roots of the polynomial with no input from the user besides the polynomial itself. The second group of algorithms all required the user to input an estimate of where the first root was located, as a result they were eliminated.

The last group of algorithms to be evaluated included Mueller's method and Bairstow's method. Bairstow's method was chosen because the algorithm did not require the user to estimate the location of the first

root, and the algorithm itself is fairly straight forward. Mueller's method also had the same benifits but was less likely to converge on the roots (12). Bairstow's is an iterative method which involves finding the quadratic factors of the polynomial (12).

After the roots have been calculated the roots and the polynomial are stored in the proper location in 'tf&pols.dat'. The defined polynomial is then displayed on the screen from memory, this allows the user to validate the input.

Matrix Input

If the user chooses to define a matrix, the define routine requests the number of rows and the number of columns of the matrix. In response to the prompts the user should enter an integer. If the integers are less than one or greater than the maximum number of rows or maximum number of columns an error message as well as the desired range of input is displayed on the screen, and the user is asked to re-input the number. At the present time the maximum number of rows and columns are set to ten. Once the number of rows and the number of columns have been entered, the screen is cleared, the left bracket for the matrix is drawn on the screen along with labels for the rows and columns. If there are five or less columns, a small matrix, the right bracket is also drawn on the screen. If the matrix has more than five columns, a large matrix, only the first five columns are displayed on the screen.

A twelve space highlighted area will appear in the (1,1) location in the matrix and a prompt will appear at the bottom of the screen. In response to the prompt the user enters a real number, the entry may be made in exponential notation or may be a string of numbers. Once a

carriage return is entered the routine checks to see if this is a valid real number. If it is valid, the number is converted to floating point notation and displayed at the proper location on the screen. The highlighted area is then moved to the next location expecting input, ICECAP-PC will fill the matrix by column. This process is repeated until the entire matrix is populated. If this is a large matrix as soon as the first five columns of the matrix are populated the routine then displays the second screen of the matrix and prompts the user for input. If at any time the user wishes to abort or exit from this option the user should enter a dollar sign, $, followed by a carriage return.

After all of the entries of the matrix are filled, the matrix is stored in the proper location in 'matrix.dat'. The defined matrix is then displayed to the user from memory, this allows the user to validate his input.

4.6.4 Display

The DISPLAY command is used to write information onto the terminal screen. This information generally takes the form of plots of system response, listings of the system specifications, or a root locus plot. DISPLAY may also be used to display the current contents of 'tf&pols.dat' and 'matrix.dat'. After the user selects the DISPLAY option the user is presented with a sub-menu, Figure 4.13.

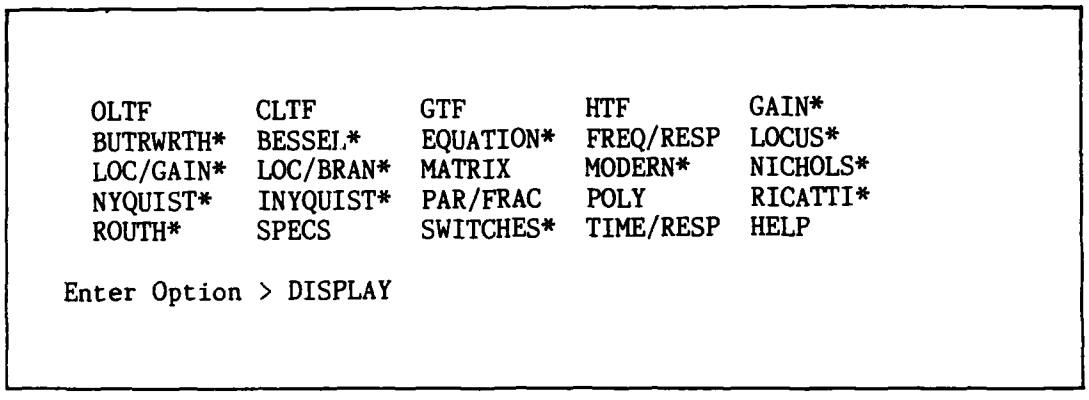


Figure 4.13: Display Command Sub-menu

All of the sub-menu options with asterisks (*) have not been implemented. Each of the implemented options will be discussed in detail. The options that have completed the design stage will also be discussed.

DISPLAY OLTF, CLTF, GTF, and HTF

After the user selects one of these four options, the routine will obtain the transfer function information from the correct location in 'tf&pols.dat'. This information is then displayed on the terminal screen. In order to limit the number of options in this large menu the other five transfer functions, TF1, TF2, TF3, TF4, and TF5, may not be directly displayed with this option. If the user desires to display these transfer functions he must first COPY them into one of the displayable transfer functions.

DISPLAY FREQ/RESP

This option provides a tabular listing of the frequency response

values over a specified range of frequencies. At the present time only the continuous time response capabilities have been implemented.

In response to the DISPLAY FREQ/RESP option the user is presented with a menu of available transfer functions. After the user has entered a valid transfer function name the routine asks the user for the initial, the final, and the delta frequency in hertz or radians per second. Whether the input frequencies are entered in hertz or in radians per second, whether the output for magnitude response is linear or in decibels, and whether the output for phase angle is in degrees or radians depends on the answers given by the user to questions posed by the routine. Based on those answers and the user input the routine will then calculate the frequency response and display a screenful of data. If there is more than one screen of data the routine will pause and ask the user to enter a carriage return when ready for more data. This routine prevents invalid input and may be aborted by entering '$' in response to any prompt.

DISPLAY LOCUS

This option computes all of the branches of the open-loop transfer function (OLTF) over a specified region of the complex plane. The process is started at each pole location, the angle of departure is determined and a point sigma distance away from the pole is calculated. The sum of the angles is calculated at this new point and the factor of pi is subtracted. This insures the sum of the angles would be zero degrees instead of an odd multiple of 180 degrees. If the sum of the angles at this new point is within the pre-established tolerance and epsilon, the point is assumed to be on the locus and the slope of the curve at that point is determined using a formula contained in TOTAL and

ICECAP (13, 10). If the sum of the angles is not within the required tolerance, a correction angle is calculated and a new point is evaluated. This cycle continues until the sum of the angles is within the tolerance. Once the locus point has been determined and the slope at that point calculated a new trial point is established and the process continues.

After each point is found, three tests are performed. First the point is evaluated to see if it lies within one sigma of a zero. If it does, it is assumed the branch terminates at that point. The second test to see if the point is outside the user selected boundaries. If it is, the branch calculation is complete. And lastly, the slope of the point is compared to the slope of the previous point. If the difference is greater than one-third pi, this branch is in a breakaway condition, the breakaway routines are entered and the actual breakaway point and departure angle is determined.

After all branches have been calculated, a border search is conducted along the user defined boundaries. If a point of the locus is found, its slope is calculated, if the slope indicates the branch will re-enter the region, the branch is followed until it meets the criteria of one the three tests above. (10, 13, 20)

In response to the DISPLAY LOCUS option the user is presented with a menu of available transfer functions. After the user has entered a valid transfer function name, the routine asks the user for area in which the root locus should be calulated. Once ICECAP-PC has a valid area it calculates the branches of the root-locus. The output is a tabular listing, the routine will display the first page of the listing to the user. If there is more than one page of output the routine will

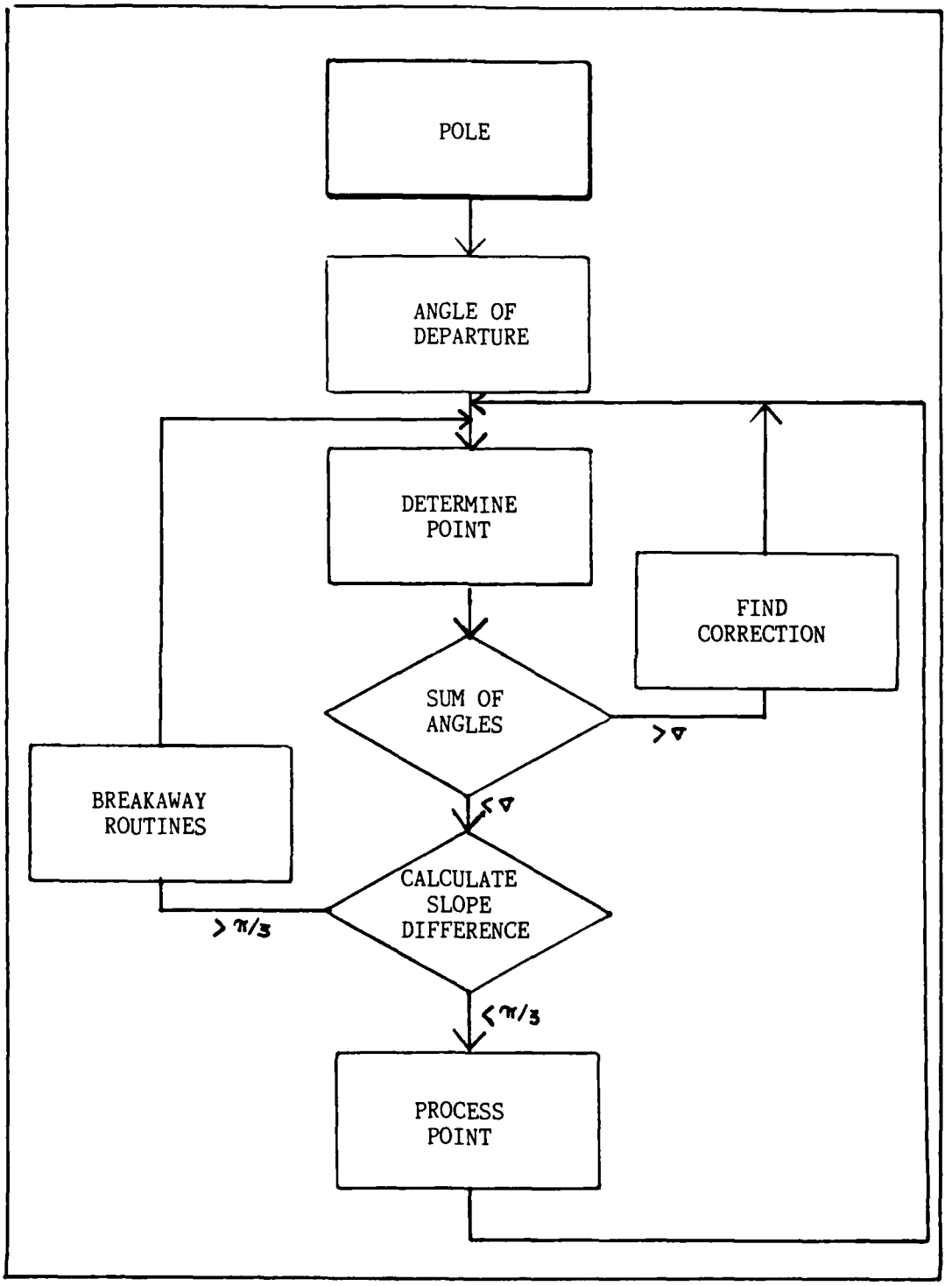


FIGURE 4.14: Root-locus Algorithm

wait for the user to enter a carriage return before displaying more data. This routine prevents invalid input and may be aborted by entering '$' in response to any prompt.

This option has been designed but has only been partially implemented.

DISPLAY LOC/GAIN

This option calculates the exact values of the closed-loop roots at the value of GAIN specified. In response to the DISPLAY LOC/GAIN option the user is presented with a menu of available transfer functions. After the user has entered a valid transfer function name the routine asks the user for the value of GAIN the roots should be calculated for. The user should input a real number in response to the prompt. Once the routine has a valid real number it calculates the closed-loop roots and displays them to the user. This routine prevents invalid input and may be aborted by entering '$' in response to any prompt.

This option has been completely designed but has only been partially implemented.

DISPLAY LOC/BRAN

This option allows the investigation of a portion of the locus, only one branch of the locus will be tabulated, beginning at a user specified point. In response to the DISPLAY LOC/BRAN option the user is presented with a menu of available transfer functions. After the user has entered a valid transfer function name, the routine asks the user for the starting point of the branch. The user should input a real number in response to the prompt for the X-location and again for the Y-location. Once the routine has valid real numbers for the starting

point it calculates the branch of the root locus and displays the first page of the tabular listing to the user. If there is more than one page of output the routine will wait for the user to enter a carriage return before displaying more data. This routine prevents invalid input and may be aborted by entering '$' in response to any prompt.

This option has been completely designed but has only been partially implemented.

DISPLAY MATRIX

If the user selects the MATRIX options the following sub-menu will be displayed to the user, Figure 4.15.

```
ADD         SUBTRACT    MATXMULT    SCLRMULT    INVERSE
TRANSPOSE   MATA        MATB        MATC        MATD
MATE        HELP

Enter Option > DISPLAY MATRIX
```

Figure 4.15: Display Matrix Command Sub-menu

The relationship and flow between each of the DISPLAY MATRIX commands is illustrated in Figure 4.16.

The ADD, SUBTRACT, and MATXMULT will add, subtract, or multiply two matrices together and store the result in a user selected location. Once the user selects one of these three options ICECAP-PC prompts the user for three matrix names. The routine will check to see is if the input matrix names are valid, if not the routine will provide an error message

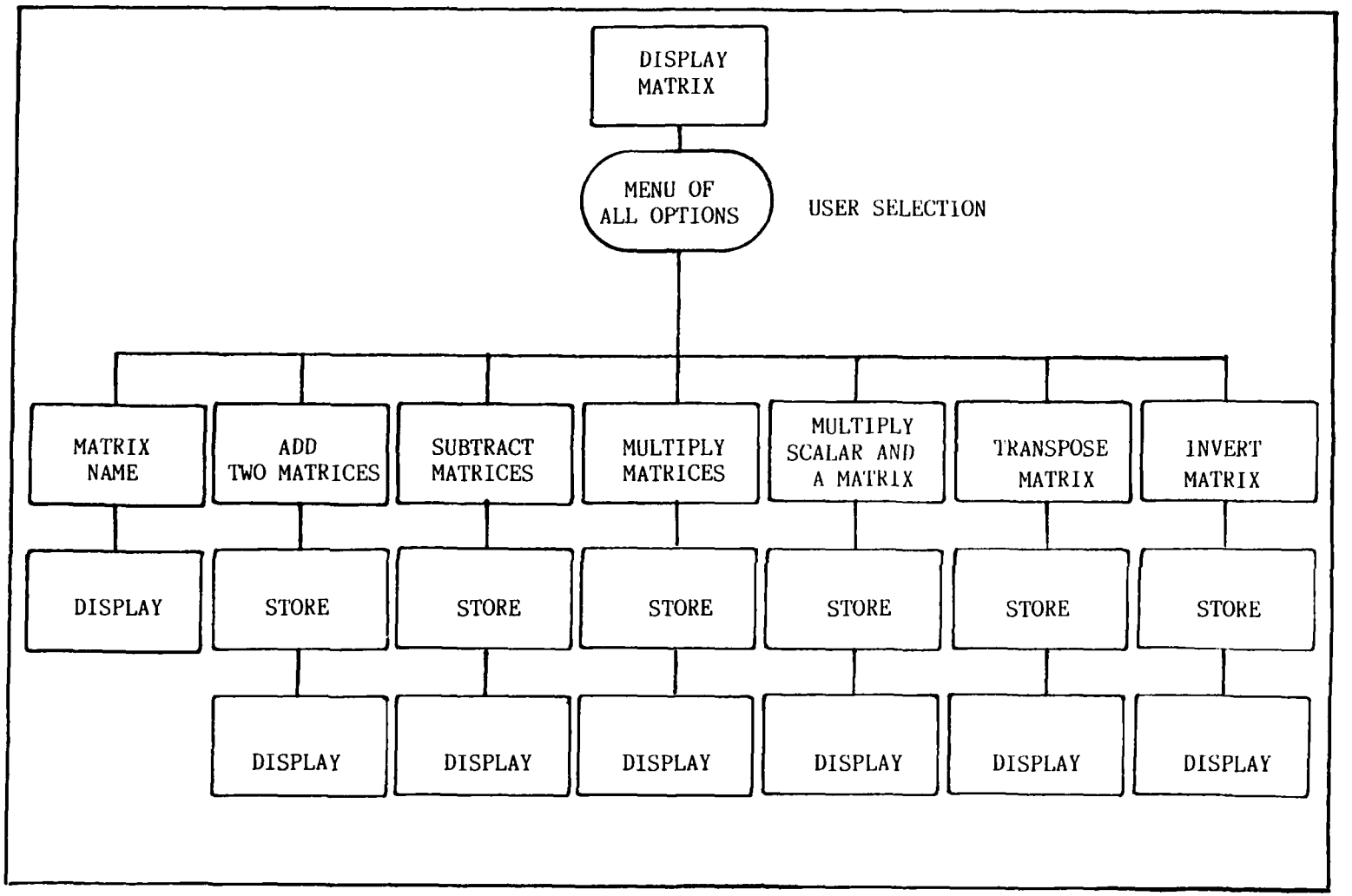


FIGURE 4.16: Display Matrix Command

and prompt the user to re-input the matrix name. If the user desires MATA may be added to MATA and the result stored in MATA.

The INVERSE command will invert a matrix and store the resulting matrix in a user selected location. The routine will check to see if the selections are valid, provide an error message is they are not, and prompt the user to re-input the matrix name. If the user desires both the inverted matrix and the storage matrix may be the same.

The TRANSPOSE command will transpose a selected matrix and store the resulting matrix in a user selected location. The routine will check to see if the selections are valid, provide an error message is they are not, and prompt the user to re-input the matrix name. If the user desires both the transposed matrix and the storage matrix may be the same.

The SCLRMULT command will multiply a matrix by a scalar and store the result in a user specified location. The user will be prompted for the multiplication matrix name and the real number that will be multiplied with all of the matrix entries. The user will also be prompted for the storage matrix name. The routine will check the validity of the matrix names as well as the real number. The routine will perform the operation and store the resulting matrix in the user selected location. If the user desires, the original and the storage matrix may be the same.

If the user enters one the five matrix names, the routine will retrieve the matrix information from 'matrix.dat' and display that information on the screen.

DISPLAY MODERN

This option allows the user to perform Modern Control Theory system

design and analysis. This option includes a Ricatti equation module. After the user enters the DISPLAY MODERN option the user is presented with a menu of available matrices. The user is then asked which matrix should be placed in which location in the Ricatti equation. After the user enters a valid matrix names in all of the required locations the routine performs the necessary calculations and displays the result to the user. The Ricatti equation module makes extensive use of matrix manipulation routines. This routine prevents invalid input and may be aborted by entering '$' in response to the prompt.

This routine has been designed but has not been implemented.

DISPLAY PAR/FRAC

This option performs the partial fraction expansion of a transfer function. After the user enters the DISPLAY PAR/FRAC option the user is presented with a menu of available transfer functions. Once the user enters a valid transfer function name the routine calculates the partial fraction expansion and displays the result to the user. This routine prevents invalid input and may be aborted by entering '$' in response to the prompt.

DISPLAY POLY

If the user selects the POLY options the following sub-menu will be displayed to the user, Figure 4.17.

```
ADD         SUBTRACT    POLYMLT     SPOLYMLT    POLYA
POLYB       POLYC       POLYD       POLYE       ONPOLY
ODPOLY      CNPOLY      CDPOLY      GNPOLY      GDPOLY
HNPOLY      HDPOLY

Enter Option > DISPLAY POLY
```

Figure 4.17: Display Poly Command Sub-menu

The relationship and flow between each of the DISPLAY POLY commands is illustrated in Figure 4.18.

The ADD, SUBTRACT, and POLYMLT will add, subtract, or multiply two polynomials together and store the result in a user specified polynomial. Once the user selects one of these three options he is prompted for the three polynomial names. The routine will check to see if the input polynomial names are valid, if not, provide an error message, and prompt the user to re-input the polynomial name. If the user desires he may add POLYA to POLYA and store the result in POLYA. Once the routine has valid polynomial names the polynomial information is retrieved from 'tf&pols.dat'. The routine will then check to see if the polynomial constant, gain is unity. If it is not, all of the coefficients of the polynomial are multiplied by the non-unity gain. The routine will then perform the desired operation. The routine will then check the resulting polynomial for a unity leading coefficient, if the leading coefficient has any value other than one, the gain/constant is multiplied by that value, all of the other polynomial coefficents are

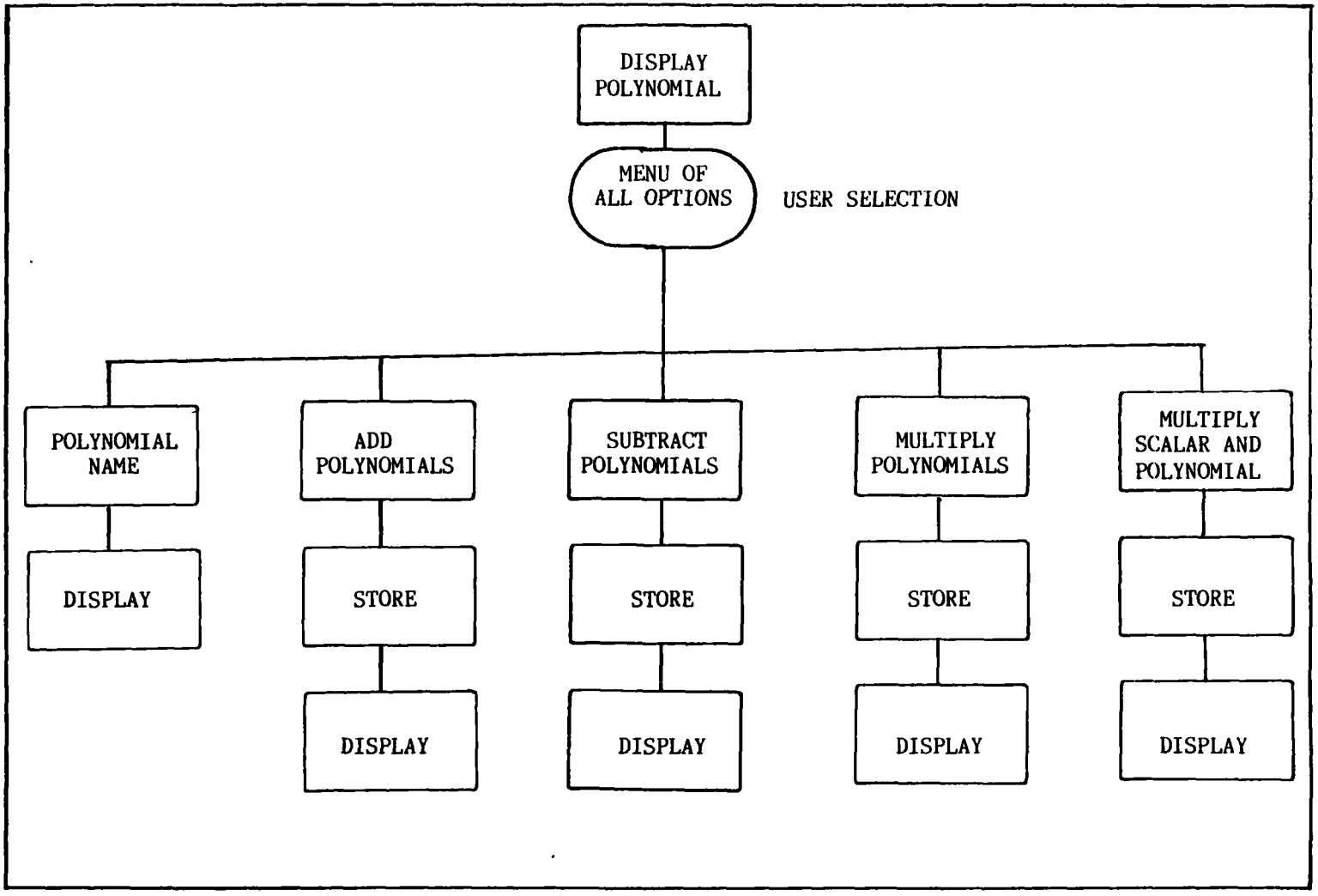


FIGURE 4.18: Display Polynomial Command

divided by the leading coefficient, and the leading coefficient is set to one. This polynomial is then stored in the user selected location in 'tf&pols.dat'.

The SPOLYMLT command multiplies a polynomial with a user selected real number and stores the resulting polynomial in a user specifed location. Once the user selects this option he is prompted for original polynomial, a real number and storage polynomial. The routine will check to see is if the input polynomial names are valid, if not provide an error message, and prompt the user to re-input the polynomial name. This routine will also check to see if the number entered in response to the real number prompt is valid. If the user desires he may store the resulting polynomial in the original polynomial location in 'tf&pols.dat'. Once the routine has valid polynomial names the polynomial information is retrieved from 'tf&pols.dat'. The routine will then perform the desired operation. The routine will then check the resulting polynomial for a unity leading coefficient. If the leading coefficients is any value other than one, the gain/constant is multiplied by that value, all of the other polynomial coefficents are divided by the leading coefficient, and the leading coefficient is set to one. This polynomial is then stored in the user selected location in 'tf&pols.dat'.

If the user enters one the thirteen polynomial names, the routine will retrieve the polynomial information from 'tf&pols.dat' and display that information on the screen.

DISPLAY SPECS

This option computes the time response characteristics that provide

a basic measurement of system performance. These characteristics include:

- Rise Time (the time to go from ten percent to ninty percent of the final value)
- Duplication Time (the time from zero to the first intersection with the final value)
- Peak Time (the time to reach the highest peak)
- Settling Time (last time at which the response was outside a two percent envelope around its fianl value)
- Peak Value (magnitude of the highest peak)
- Final Value (value of response as time approaches infinity)

After entering DISPLAY SPECS option the user is presented with a menu of available transfer functions. Once the user enters a valid transfer function name the routine calculates the 'specs' and displays them to the user. This routine prevents invalid input and may be aborted by entering '$' in response to the prompt.

DISPLAY TIME/RESP

This option provides a tabular listing of the time response values over a specified range of time. At the present time only the continuous time response capabilities have been implemented. After the user selects DISPLAY TIME/RESP option the user is presented with a menu of available transfer functions. After the user has entered a valid transfer function name the routine asks the user for the initial time, the final time, and the desired time increment. The routine will then calculate the time response and display a screenful of data. If there is more than one screen of data the routine will pause and ask the user to enter a

carriage return when ready for more data. This routine prevents invalid input and may be aborted by entering '$' in response to any prompt.

4.6.5 Form Command

The FORM command does not have any lower level menus per se, instead it presents the user with a list of different types of the form command and then asks the user to input the integer value of his choice, Figure 4.19.

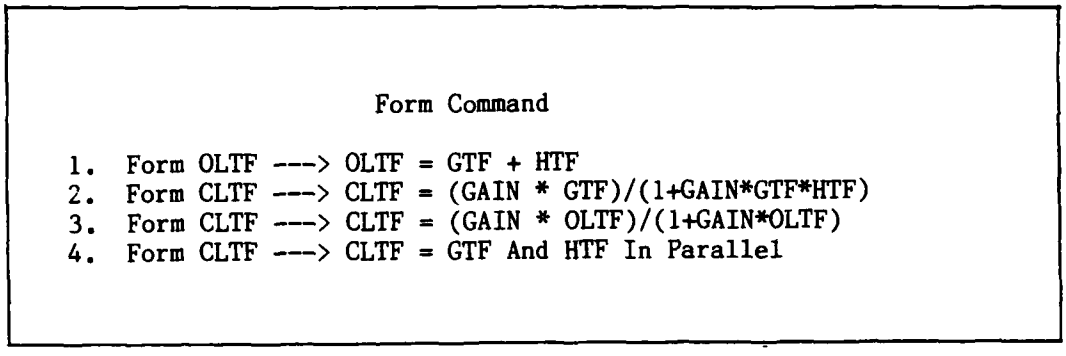


Figure 4.19: Form Command

The routine will execute the user selection. After the operation is complete the selection is stored in the proper location. The formed transfer function is then displayed to the user so that he may validate the results.

4.6.6 Help Command

The help functional area provides on-line assistance to the user. After the user inputs the help command ICECAP-PC responds by presenting the help submenu, Figure 4.20.

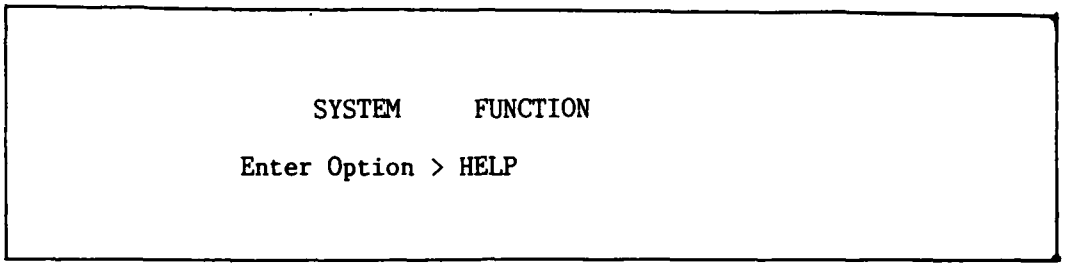


Figure 4.20: Help Command Submenu

HELP SYSTEM

If the user selects the SYSTEM option, the help routine determines the location of the information in the HELP.SYS file by referencing a message directory. The screen is then cleared, and the text displayed. The routine will display a screenful of data and then ask the user to input a carriage return when he wishes to see the rest of the message. The user may abort the help sequence any time the routine is waiting for user response by inputting a dollar sign, $, followed by a carriage return.

HELP FUNCTION

If the user selects the FUNCTION option a sub-sub-menu will be presented to the user, Figure 4.21.

```
CHANGE    DISPLAY   MODIFY    RECOVER   SWITCHES
COPY      FORM      PRINT     STOP      UPDATE
DEFINE    HELP

Enter Option > HELP FUNCTION
```

Figure 4.21: Help Function Sub-sub-menu

This menu is the same as the main menu presented to the user after initialization and after each operation. From this menu the user may get on-line help for any of the twelve main areas. After the user selects an option, the help routine determines the location ot the information in the HELP.SYS file by referencing a message directory. The screen is then cleared, and the text displayed. The routine will display a screenful of data and then ask the user to input a carriage return when he wishes to see the rest of the message. The user may abort the help sequence any time the rouitne is waiting for user response by inputing a dollar sign, $, followed by a carriage return.

4.6.7 Modify Command

The modify command is used to change polynomials and matrices without redefining the entire polynomial or matrix. This command is limited to changing a polynomial or matrix. If the user desires to change a transfer function he may modify either the numerator polynomial, to add or delete a zero, or the denominator polynomial, to

add or delete a pole, or both. After the user inputs the modify command a submenu is presented to the user, Figure 4.22.

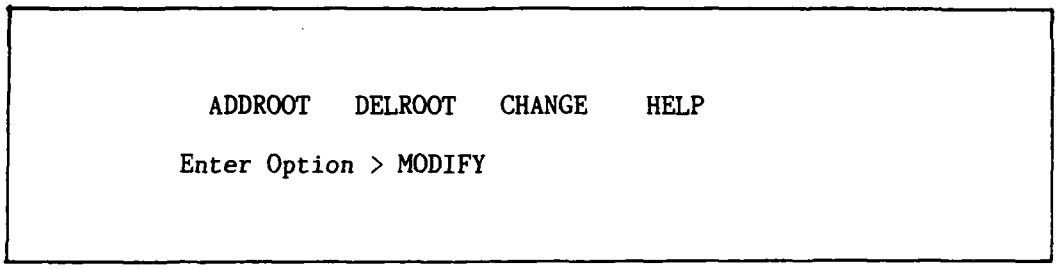


Figure 4.22: Modifiy Sub-menu

The relationship between and the flow in each of these options is illustrated in Figure 4.23.

MODIFY ADDROOT

ADDROOT is used to add a root to a polynomial. In response to the ADDROOT command ICECAP-PC will provide a menu of available polynomials. After the user has correctly entered the name of a polynomial, ICECAP-PC will display the polynomial and highlight the real portion of the new root. In response to the prompt at the bottom of the screen the user should enter a real number. After the real number is correctly entered the highlighted area moves to the imaginary part of the root, if any input other than zero is made ICECAP-PC will calculate the conjugate of the root. After the root has been correctly entered ICECAP-PC will recompute the polynomial and store the result in 'tf&pols.dat'.

MODIFY DELROOT

DELROOT is used to delete a root from a polynomial. In response to the DELROOT command ICECAP-PC will provide a menu of available polynomials. After the user has correctly entered the name of the

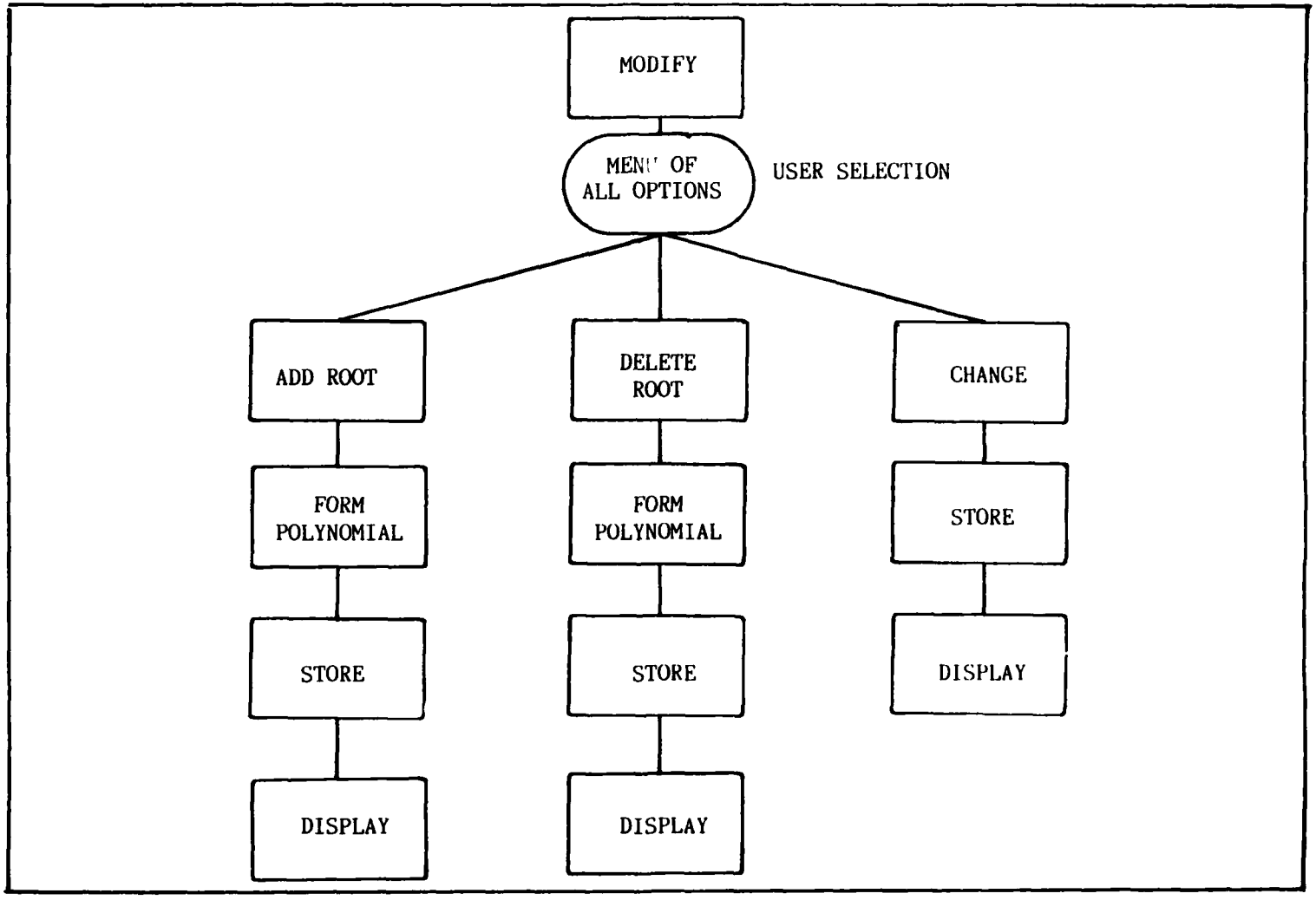


FIGURE 4.23: Modify Command

polynomial the routine will ask the user for the number of the root he wishes to delete. The roots are numbered on the left of the factored display. The root number must be an integer and be available for elimination. If the root the user has selected is complex the conjugate is also deleted. The routine will then recompute the new polynomial and store the result in 'tf&pols.dat'.

MODIFY CHANGE

The CHANGE command is used to modify a single location in a matrix. In response to the CHANGE command ICECAP-PC will provide a menu of matrices that may be modified. After the user has correctly entered the name of the matrix, the matrix is displayed and the user is asked for the row and column location of the modification. The routines are written so that only those columns on display may be altered. If the modification matrix is large, more than five columns, and the location the user desires to modify is on the second page of the display he should enter 'next' in respose to the row number prompt. The second page of the matrix will appear, and the user will be prompted for the row and column location. After the modification has been made ICECAPP-PC will store the matrix in the proper location in 'matrix.dat'.

4.6.8 Print

The PRINT command is used to write information to an external file in addition to displaying the information on the screen. The contents of the external file may then be sent to the printer at the conclusion of the session. ICECAP-PC will provide a list of allowable objects in response to the PRINT command, Figure 4.24.

```
OLTF*       CLTF*       GTF*        HTF*        GAIN*
BUTRWRTH*   BESSEL*     EQUATION*   FREQ/RESP*  LOCUS*
LOC/GAIN*   LOC/BRAN*   MATRIX*     MODERN*     NICHOLS*
NYQUIST*    INYQUIST*   PAR/FRAC*   POLY*       RICATTI*
ROUTH*      SPECS*      SWITCHES*   TIME/RESP*  HELP

Enter Option > PRINT
```

Figure 4.24: Print Command Sub-menu

As indicated by the asterisks this option has not been implemented.

4.6.9 Recover Command

The recover command does not have any lower level menus. This function allows the user to write the contents of user specified files into ICECAP-PC files, 'tf&pols.dat' and 'matrix.dat'. The routine prompts the user for the names of the user specified files. The user is cautioned not to put the user specified file for transfer functions and polynomials into 'matrix.dat' or vice versa. Equally, the user specified files must exist on the disk.

4.6.10 Stop Command

The STOP function is is designed so that the user may exit from ICECAP-PC gracefully. Information is always stored in the disk files 'tf&pols.dat' and 'matrix.dat' for later use. Exiting from ICECAP-PC does not alter these disk files. The STOP command is the 'normal' mode for exiting ICECAP-PC.

4.6.11 Switches Command

The SWITCHES function has been designed but not implemented and does not have any lower level menus. This function will allow the user to change or flip the boolean switches in ICECAP-PC.

4.6.12 Update Command

The update command does not have any lower level menus. This function allows the user to write the contents of the ICECAP-PC files 'tf&pols.dat' and 'matrix.dat' to user specified files. This command is particularly useful when there is more than one user of ICECAP-PC or a single user is working on more than one design in parallel. The limitations on the file names are those placed on the user by the operating system. In the case of MS/PC-DOS the user specified file name must be eight characters or less, with no spaces between the characters. The user specified file will overwrite disk file if it has the same name. If the file does not exist on the disk ICECAP-PC will create it.

4.7 Memory Management

In order to minimize the the execution time of ICECAPPC memory management techniques were avoided until the end of the Implementation Phase. Without memory management techniques the selected design language, TURBO-Pascal will compile approximately 8000 lines of code into the 64K allocated memory. When this barrier was reached several of the procedures were placed into overlays. The overlay procedures are listed in Appendix E. Overlays return control to the main program after they have executed, the other memory management alternative, chaining, did not.

4.8 Summary

This chapter discussed the detailed design of ICECAP-PC and its implementation. Specifically, the major modifications made to MICROSDW described along with the difficulties discovered while translating the detailed design into TURBO-Pascal. The design approach taken during the development of the user interface and the twelve functional areas was a 'top-down' design approach. This approach facilitated the modular design of ICECAP-PC, and allowed error protection/detection to be designed into the user interface code.

Chapter V will discuss the testing of the implemented design. It is important to remember that the design, implementation, and test of a software system is an iterative process.

CHAPTER V

TESTING AND VALIDATION

5.1 Introduction

Throughout all six phases of the software life cycle, testing and validation is performed. The purpose of software testing is twofold. First, the designer must test each of the subroutines to determine if they are performing their desired function correctly. Are the calculations accurate? Is the data being moved or stored properly? Second, the designer must validate the software with the requirements laid out in the Requirements Definition phase. Is the program meeting the needs of the user? Have the needs changed? The products of each phase in the software life cycle must be continually compared with the user requirements. It is through this recurring testing and validation that errors are discovered before they become too far-reaching.

This chapter is divided into two major sections, Validation and Testing. The Validation section takes the twelve functional areas of the system software (as described in Chapter 4, Implemented Design), and compares them with the system requirements (described in Chapter 2, Requirements Definition). This validation process ensures that the program is meeting the requirements, and therefore, the needs of the user. The Testing section of this chapter determines whether the mathematical subroutines are performing their desired calculations correctly. Answers are compared with the output of similar routines using other CAD packages on larger computers. If the routines are simple enough, answers are checked by doing calculations by hand, or by checking values in standard mathematical tables. For the purposes of

this thesis effort, the test method used was the one determined to be most appropriate for the specific procedure being tested at the time.

By testing against known values and validating against system requirements, it can be determined if the software is performing properly. To begin that determination, validation against system requirements is looked at first.

5.2 Validation

Validation is the process of comparing the implemented design of a software project to the requirements, to ensure the program meets the needs of the user. ICECAP-PC's design is structured into twelve functional areas:

Change	Form	Recover
Copy	Help	Stop
Define	Modify	Switches
Display	Print	Update

Each of these areas are covered in detail in separate sub-sections. The overall philosophy for validation is simple. First, a general description of the functional area is presented. Then, specific criteria for validating the functional area against its requirements is determined. Finally, the criteria are applied to each area, and the results are documented for future reference.

5.2.1 General Criteria

Before starting on the validation criteria for each of the twelve functional areas, some general rules must be stated first. These rules are applicable to all twelve functional areas, so they are referred to as 'system criteria'. The system criteria are presented below.

* The user must be able to exit any option or function by entering the '$' abort command any time the system prompts for user input.

* Any incorrect entry by the user must generate an error message which prompts the user to re-enter the input.

These system criteria have been consistently met throughout the program. Now that the system criteria are defined, it is time to discuss the criteria for each of the twelve functional areas.

5.2.2 Change

The Change command allows the user to select a different plane for analysis or a different sampling time. The criteria for validating this functional area is the following:

* When changing analysis plane or sampling time, the system displays the current plane or time before listing the choices.

* If the user desires to change the analysis plane, the system provides a list of planes to choose.

* If the user desires to change the sampling time, the system provides a list of appropriate times to choose.

* Once the user makes a choice, the system displays the new plane or sampling time.

Since the Change option only reached the design stage, validation could not be performed. However, when the option is finally implemented, the above criteria must apply.

5.2.3 Copy

The Copy function allows the user to copy transfer functions, polynomials, or matrices from one storage location to another. The criteria for validating this functional area is the following:

* When entering the Copy option, the system provides the user with a list of transfer functions, polynomials, or matrices that are available as sources.

* If the user selects a transfer function as a source, the system will provide a list of appropriate transfer functions for destinations.

* If the user selects a polynomial as a source, the system will provide a list of appropriate polynomials for destinations.

* If the user selects a matrix as a source, the system will provide a list of appropriate matrices for destinations.

* When performing a copy, the system displays a message showing the source and destination of the copy.

All of the above criteria were used to validate the Copy function.

Results: All criteria were successfully met.

5.2.4 Define

The Define command allows the user to input transfer functions, polynomials, and matrices. The criteria for validating this functional area is the following:

* When entering the Define option, the system displays a list of appropriate transfer functions, polynomials, or matrices for the user to choose.

* If the user desires to input a transfer function or polynomial, the system gives the user a choice of entering it in factored or polynomial form.

* If the user chooses to input a transfer function, the system prompts for the degree of the numerator and denominator.

* If the user chooses to input a polynomial, the system prompts for the degree of the polynomial.

* If the user chooses to input a matrix, the system prompts for the number of rows and columns.

* If the user enters a degree greater than ten for a transfer function or polynomial, the system displays an error message and requests the user to re-enter the input.

* If the user enters a number greater than ten for the number of rows or columns in a matrix, the system displays an error message and requests the user to re-enter the input.

* When the user inputs a transfer function or polynomial in polynomial form, the system displays the polynomial form, factors it, then also displays the factored form.

* When the user inputs a transfer function or polynomial in factored form, the system displays the factored form, multiplies the factors, then also displays the polynomial form.

* The system must accept real numbers as input for polynomial coefficients, polynomial factors, or matrix elements.

* When entering a complex root, the system automatically assumes the complex conjugate, displays it, and notifies the user.

* If the user attempts to enter a complex root for the last factor, the system does not accept it and notifies the user that there is no room for the conjugate pair.

The above criteria were used to validate the Define function. Results: All criteria were successfully met.

5.2.5 Display

The Display command allows the user to display transfer functions, polynomials, and matrices. It also provides features for manipulating polynomials and matrices, and provides analysis tools such as frequency response, time response, root locus, etc. The following criteria is used to validate this functional area:

* When entering the Display option, the system provides the user with a list of transfer functions, polynomials, and matrices that are available for display.

* When entering the Display option, the system also provides the user with a list of available analysis tools, and a list of polynomial and matrix manipulation features.

* The format of transfer functions, polynomials, and matrices in the Display option are the same as the corresponding formats in the Define option.

* If the user selects an analysis tool or a manipulation feature, the appropriate routine is found and executed by the system.

The above criteria were used to validate the Display function. Results: All criteria were successfully met.

5.2.6 Form

The Form command allows the user to form an OLTF or CLTF from other transfer functions. The following criteria is used for validating this functional area:

* When entering the Form option, the system provides the user with a list of methods for forming the OLTF or CLTF.

* If the user selects one of the methods, the system performs the calculations and then displays either the OLTF or CLTF, depending on which one was selected.

The above criteria were used to validate the Form function. Results: All criteria were successfully met.

5.2.7 Help

The Help command is actually an entire sub-system which provides on-line assistance to the user. The following criteria is used to validate this functional area:

* When entering the Help option, the system provides the user with a choice of general system level help or specific function level help.

* The Help sub-system must provide specific help on all twelve major functional areas.

* If the help message fills more than the entire screen, only the first page is displayed, and the user is given the option to view more or abort.

The above criteria were used to validate the Help function. Results: All criteria were successfully met.

5.2.8 Modify

The Modify command allows the user to change polynomials or matrices without redefining the entire entity. The following criteria is used to validate this functional area:

* When entering the Modify option, the system provides the user with a choice of changing polynomials or matrices.

* A polynomial can be changed by adding or deleting a root.

* A matrix can be changed by modifying a single element.

* If the user chooses to modify a polynomial, the system provides a list of available polynomials.

* If the user chooses to modify a matrix, the system provides a list of available matrices.

* When modifying a polynomial, the system displays the appropriate polynomial and prompts the user for the root to be added or deleted.

* When modifying a matrix, the system displays the appropriate matrix and prompts the user for the row and column of the element to be changed.

* When a polynomial or matrix is changed, the new form is displayed to the user for verification.

The above criteria were used to validate the Modify function. Results: All criteria were successfully met.

5.2.9 Print

The Print command allows the user to send information to an external file for later output to a printer. The only difference between the Print and Display command is that Display sends information to the screen only, while Print sends it to the screen and to an external file. Because these two commands are so similar, the same criteria used for validating Display is also used for Print. Print, however, has some additional criteria as listed below.

* When using the Print command, the system notifies the user of the name of the file where the information is being sent.

* If a print file is established by using the Print command, then when the user terminates the session, the system reminds the user that a print file was created.

Since the Print option only reached the design stage, validation could not be performed. However, when the option is finally implemented, the above criteria apply.

5.2.10 Recover

The Recover command allows the user to write the contents of user specified files into the system data files. The following criteria is used to validate this functional area:

* When entering the Recover option, the system prompts the user for the names of the user specified files.

* The system provides a warning to the user not to recover polynomial data files into matrix data files or vice versa.

The above criteria were used to validate the Recover function. Results: All criteria were successfully met.

5.2.11 Stop

The Stop command is the normal way for the user to exit the system. The following criteria is used to validate this functional area:

* When the user enters the Stop command, the screen clears and the operating system prompt appears.

* If any file messages, such as print files, need to be displayed, they are provided before the operating system prompt appears.

Since the Print command is not implemented yet, there is no requirement to display any messages after the user enters Stop. Therefore, the second criteria could not be used for validating the Stop command. However, the first criteria did apply and was successfully met.

5.2.12 Switches

The Switches command allows the user to set system level boolean switches. The following criteria is used to validate this functional area:

* When entering the Switches option, the system provides the user with a list of the system switches and their current value.

* The system prompts the user to enter a switch and then enter if it is 'off' or 'on'.

Since the Switches option only reached the design stage, validation could not be performed. However, when the option is finally implemented, the above criteria apply.

5.2.13 Update

The Update command allows the user to write the contents of the system data files to user specified files. The following criteria is used to validate this functional area:

* When entering the Update option, the system prompts the user for the names of the user specified files.

* If the files that the user specified do not exist, then the system will create them.

The above criteria were used to validate the Update function. Results: All criteria were successfully met.

5.3 Testing

Testing is the process of determining if the subroutines are performing their desired functions correctly. Of particular interest are the routines that perform mathematical calculations. ICECAP-PC has dozens of them. The general procedure for testing these routines is to use input values which already have known output. The answers calculated by the routine can then be compared with the known output. Mathematical tables, hand calculations, book examples, and other computer aided design packages are all sources for known values of input and output. Testing of ICECAP-PC subroutines involved all of these sources. Specific test results for a number of different subroutines are presented here in separate subsections. The following is a list of procedures whose testing results are included in this chapter:

arc_tangent
log10
frequency_response
magnitude
phase_angle
roots

5.3.1 Arc tangent

The function arc_tangent was developed because TURBO Pascal's arctan function only returns a value between -pi/2 and pi/2. Because arctan is limited to quadrants one and four, arc_tangent was created to

handle coordinates in all four quadrants. This new function returns the angle in radians, with a range of -pi to pi (where zero is the positive x-axis).

To test arc_tangent, x and y values for all four quadrants (including the x and y axes) were input. The results were then compared with output from the double precision FORTRAN function DATAN2. DATAN2 results were obtained from both the VAX 11/780 and CYBER computers. A total of 121 test points were run. Selected points are shown in Table 5.1.

TABLE 5.1

TEST RESULTS FOR FUNCTION ARC_TANGENT

x	y	Results in Radians ICECAP-PC	VAX and CYBER
4	0	0.0000000000	0.00000000000000
4	1	.24497866313	.244978663126864
4	4	.78539816340	.785398163397448
1	4	1.3258176637	1.32581766366803
0	4	1.5707963268	1.57079632679489
-1	4	1.8157749899	1.81577498992176
-4	4	2.3561944902	2.35619449019234
-4	1	2.8966139905	2.89661399046292
-4	0	3.1415926536	3.14159265358979
-4	-1	-2.8966139905	-2.89661399046292
-4	-4	-2.3561944902	-2.35619449019234
-1	-4	-1.8157749899	-1.81577498992176
0	-4	-1.5707963268	-1.57079632679489
1	-4	-1.3258176637	-1.32581766366803
4	-4	-.78539816340	-.785398163397448
4	-1	-.24497866313	-.244978663126864

Table 5.1 does not show the full accuracy of the VAX or CYBER computers. The output was truncated 15 significant digits (i.e. 1 digit to the left of the decimal point and 14 digits to the right, or no digits on the left of the decimal point and 15 on the right). Double precision on the

VAX actually gives 16 significant digits of accuracy, with the 16th digit rounded. The CYBER has 27 digits of accuracy when working in double precision. Again, the 27th digit is rounded. For our testing purposes, the truncated values shown in Table 5.1 were more than adequate.

The test results indicate that the function arc_tangent can take an x and y coordinate and correctly determine the angle in radians. Test results also verified the range as -pi to +pi radians. As shown in Table 5.1, the accuracy of arc_tangent was 11 significant digits, with the last digit rounded.

Also tested was the catch code for x=y=0. Normally, when x and y are both zero, the arctangent cannot be calculated. However, to prevent the program from exiting to the operating system when this occurs, the value of arc_tangent is set to an abnormally high number (9.9999999999E+30). This extreme result will indicate that something is wrong without halting the entire program. Test results showed that this feature worked correctly.

5.3.2 Log10

The function log10 was developed because TURBO Pascal's only intrinsic logarithm function is for natural logs (log base e). A new routine had to be created to find logarithms in base ten. The algorithm used in log10 was taken from reference 28.

To test log10, real numbers from 0.1 to 20 were input. The results were then compared with the double precision FORTRAN function DLOG10. DLOG10 results were obtained from the CYBER. A total of 29 test points were run. Selected points are shown in Table 5.2.

TABLE 5.2

TEST RESULTS FOR FUNCTION LOG10

number	Log Base Ten ICECAP-PC	CYBER
1.0	.00000000006	0.00000000000000
2.0	.30102999568	.301029995663981
4.0	.60205999143	.602059991327962
6.0	.77815125048	.778151250383643
8.0	.90308998704	.903089986991943
10.0	.99999999999	1.00000000000000
12.0	1.0791812460	1.07918124604762
14.0	1.1461280357	1.14612803567823
16.0	1.2041199826	1.20411998265592
18.0	1.2552725051	1.25527250510330
20.0	1.3010299957	1.30102999566398

As was true in the previous table, this table also lists results from the CYBER truncated to 15 significant digits.

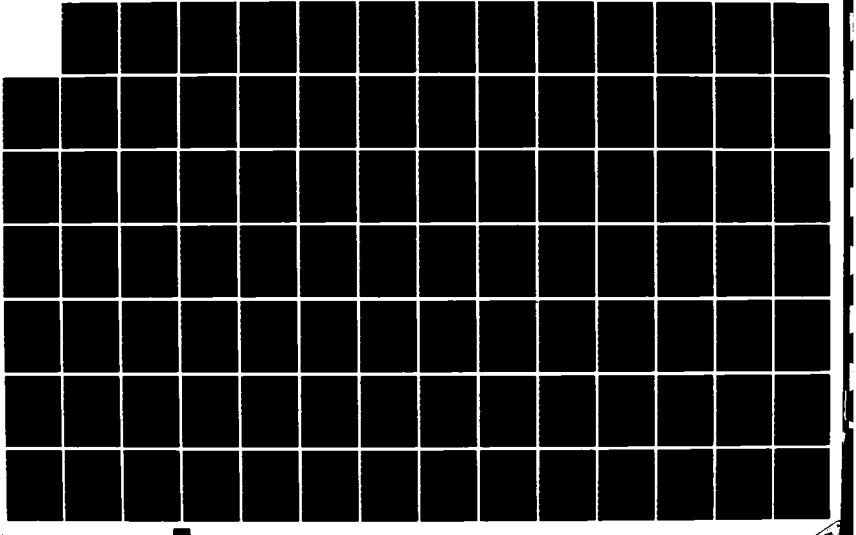
The test results indicate that for values between 0 and 10, log10 is accurate to only 9 significant digits. For numbers greater than 10, log10 is accurate to 11 significant digits, with the last digit rounded.

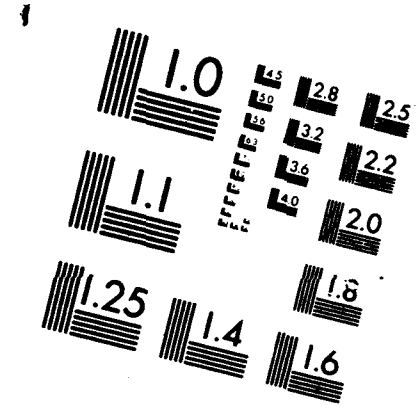
5.3.3 Frequency response

The procedure frequency_response performs a few calculations itself, but mostly, it acts as an executive. It calls all the other procedures and functions which perform calculations for frequency response analysis. There are three main functions that frequency_response calls; log10, magnitude, and phase_angle. Log10 is a stand-alone function. It can be used anywhere in the program, just like any of TURBO Pascal's intrinsic functions. Hence, the test results for log10 are presented in a separate section of this chapter. Magnitude and phase_angle, even though they are functions like log10, do not have

AD-A163 940 DEVELOPMENT OF A COMPUTER AIDED DESIGN PACKAGE FOR CONTROL SYSTEM DESIGN A.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/3

UNCLASSIFIED S K MASHIKO ET AL. DEC 85 F/G 9/2 NL





any meaningful test results unless they are considered as part of the overall frequency_response procedure. Therefore, only the results from frequency_response are presented here. It includes the calculations from both the magnitude and phase_angle functions.

To test frequency_response, the following transfer function was entered: (s+1) / (s+2)(s+3). Omega was used as the variable for frequency. For the test, omega began at a value of 0.1, and was incremented by 0.1 until it reached a final value of 10.0. This generated 100 test points. Both magnitude and phase angle were calculated at each of the test points. Also as part of the test, the units were varied; frequency from radians/second to hertz, magnitude from normalized units to decibels, and phase angle from radians to degrees. (normalized units means the maximum value for magnitude can never be more than one.) There were eight different combinations of units as shown in Table 5.3. Tests were run for all eight combinations. This produced a total of 800 test points.

TABLE 5.3

UNIT COMBINATIONS FOR FREQUENCY_RESPONSE OUTPUT

Frequency (omega)	Magnitude	Phase Angle
radians/second	normalized	radians
hertz	normalized	radians
radians/second	decibels	radians
hertz	decibels	radians
radians/second	normalized	degrees
hertz	normalized	degrees
radians/second	decibels	degrees
hertz	decibels	degrees

The results obtained from ICECAP-PC were compared with answers from two CAD packages. On the CYBER, frequency response data was obtained from TOTAL, and on the VAX, the same data was generated by ICECAP. TOTAL and ICECAP use the same routines for calculating frequency response, however, it was thought that greater accuracy could be obtained with the CYBER. Surprisingly, it was found that the tabular output from both TOTAL and ICECAP contained only 6 significant digits. ICECAP-PC produces 11. Hence the output from ICECAP-PC could only be verified up to 6 digits.

It would be much too cumbersome to include all of the test results here. Therefore, only selected values are presented in the next two tables. However, any conclusions drawn from the tables here will be indicative of the overall test results. Table 5.4 compares results for magnitude. Frequency is in hertz and magnitude is in decibels. Phase angle test results are compared in Table 5.5. Hertz is again used for frequency, and angle is in degrees. It is interesting to note that TOTAL and ICECAP-PC can calculate phase angle in degrees or radians, while ICECAP is limited to degrees only.

TABLE 5.4

TEST RESULTS FOR FUNCTION: MAGNITUDE

Frequency (hertz)	Magnitude (decibels) ICECAP-PC	VAX and CYBER
0.1	-14.713173482	-14.7132
0.2	-13.595428701	-13.5954
0.3	-13.185691739	-13.1857
0.5	-13.816131311	-13.8161
0.7	-15.122054082	-15.1221
1.0	-17.165986384	-17.1660
2.0	-22.306141667	-22.3061
3.0	-25.651073723	-25.6511
5.0	-29.995588510	-29.9956
7.0	-32.892443373	-32.8924
10.0	-35.976784981	-35.9768

TABLE 5.5

TEST RESULTS FOR FUNCTION: PHASE_ANGLE

Frequency (hertz)	Phase angle (degrees) ICECAP-PC	VAX and CYBER
0.1	2.8722947894	2.87229
0.2	-3.3815822151	-3.38158
0.3	-13.392402188	-13.3924
0.5	-31.495854331	-31.4959
0.7	-44.058998594	-44.0590
1.0	-55.863439574	-55.8634
2.0	-72.079762471	-72.0798
3.0	-77.937116980	-77.9371
5.0	-82.725715402	-82.7257
7.0	-84.796809396	-84.7968
10.0	-86.355052013	-86.3551

The results shown in Tables 5.4 and 5.5 indicate that the procedure frequency_response correctly calculates the magnitude and phase angle to at least 6 significant digits. Also, the procedure properly handles conversion of units from radians/second to hertz, from normalized units to decibels, and from radians to degrees.

5.3.4 Roots

The procedure roots was developed to find both real and complex roots of up to a tenth order polynomial. The algorithm used is Bairstow's method (ref. Applied Numerical Methods for Digital Computation). Bairstow's method is iterative, and it involves finding the roots of a polynomial by using quadratic factors.

Testing of the procedure roots is divided into two main areas, repeated roots and non-repeated roots. Like all root finders, Bairstow's method is much more successful when dealing with non-repeated roots (12).

Repeated roots

To test Bairstow's method, polynomials were generated using (s+1) as the only factor. They were input in polynomial form using the DEFINE option. The polynomial was automatically factored by the procedure roots, and then displayed next to the polynomial form. Table 5.6 shows the results.

TABLE 5.6

TEST RESULTS FOR PROCEDURE: ROOTS
(repeated roots)

Polynomial:	s + 1
Actual factors:	(s+1)
Bairstow's:	(s+1)
Polynomial:	s**2 + 2s + 1
Actual factors:	(s+1)(s+1)
Bairstow's:	(s+1)(s+1)
Polynomial:	s**3 + 3s**2 + 3s + 1
Actual factors:	(s+1)(s+1)(s+1)
Bairstow's:	(s+.999909)(s+1.00005+j.000079) (s+1.00005-j.000079)

```
Polynomial:      s**4 + 4s**3 + 6s**2 + 4s + 1
Actual factors:  (s+1)(s+1)(s+1)(s+1)
Bairstow's:      (s+1.00062)(s+1+j.00062)
                 (s+.999378)(s+1-j.00062)

Polynomial:      s**5 + 5s**4 + 10s**3 + 10s**2 + 5s + 1
Actual factors:  (s+1)(s+1)(s+1)(s+1)(s+1)
Bairstow's:      unable to find roots

Polynomial:      s**6 + 6s**5 + 15s**4 + 20s**3 + 15s**2
                      + 6s + 1
Actual factors:  (s+1)(s+1)(s+1)(s+1)(s+1)
                 (s+1)
Bairstow's:      (s+1.00062)(s+.999378)
                 (s+1+j.00062)(s+.999819+j.0085)
                 (s+1-j.00062)(s+.999819-j.0085)

Polynomial:      s**7 + 7s**6 + 21s**5 + 35s**4 + 35s**3
                      + 21s**2 + 7s + 1
Actual factors:  (s+1)(s+1)(s+1)(s+1)(s+1)
                 (s+1)(s+1)
Bairstow's:      unable to find roots

Polynomial:      s**8 + 8s**7 + 28s**6 + 56s**5 + 70s**4
                      + 56s**3 + 28s**2 + 8s + 1
Actual factors:  (s+1)(s+1)(s+1)(s+1)(s+1)
                 (s+1)(s+1)(s+1)
Bairstow's:      unable to find roots

Polynomial:      s**9 + 9s**8 + 36s**7 + 84s**6 + 126s**5
                      +  126s**4 + 84s**3 + 36s**2 + 9s
                      + 1
Actual factors:  (s+1)(s+1)(s+1)(s+1)(s+1)
                 (s+1)(s+1)(s+1)(s+1)
Bairstow's:      unable to find roots

Polynomial:      s**10 + 10s**9 + 45s**8 + 120s**7
                       + 210s**6 + 252s**5 + 210s**4
                       + 120s**3 + 45s**2 + 10s + 1
Actual factors:  (s+1)(s+1)(s+1)(s+1)(s+1)
                 (s+1)(s+1)(s+1)(s+1)(s+1)
Bairstow's:      unable to find roots
```

The test results in Table 5.6 indicate that Bairstow's method can factor exactly a polynomial with a single or two identical roots. With three and four multiple roots, complex factors appear. This is due to Bairstow's use of the quadratic equation for finding roots. The three identical roots are factored by Bairstow's method into a single root and

a complex pair. Four identical roots are factored into two distinct real roots and a complex pair. However, if the results are limited to 4 significant digits, the complex portions disappear, and the remaining real factors are still fairly accurate.

For roots of multiplicity 5, Bairstow's method is unsuccessful. With 6 identical roots, the method returns two complex pairs and two distinct real roots. However, the results must be limited to 3 significant digits to allow the complex portions to disappear. The ability of Bairstow's method to successfully factor 6 multiple roots, but not 5, again lies with its use of quadratic factoring. An even number of roots can be factored more easily than an odd number. Roots of multiplicity 7 or greater cannot be factored by Bairstow's method.

It is recommended that polynomials with no more than four identical roots be entered in polynomial form by DEFINE. This will ensure the greatest accuracy when calculating factors. As a warning, entering polynomials with roots of multiplicity 7 or greater will cause the program to abort with a floating point overflow, or will produce absurd values for factors.

Non-repeated roots

To test Bairstow's method for non-repeated roots, polynomials were generated using factors (s+1), then (s+1)(s+2), then (s+1)(s+2)(s+3), etc. The polynomials were entered in the DEFINE option and were automatically factored. Bairstow's method was able to exactly determine up to nine non-repeated roots. For instance, the polynomial s**9 + 45s**8 + 870s**7 + 9450s**6 + 63273s**5 + 269325s**4 + 723680s**3 + 1172700s**2 + 1026576s + 362880 was correctly factored into

(s+1)(s+2)(s+3)(s+4)(s+5)(s+6)(s+7)(s+8)(s+9). The only limitation on finding non-repeated roots is the accuracy of real numbers in TURBO Pascal (11 significant digits). If any of the polynomial coefficients exceed approximately 8 digits in length, then the root finder will begin to show accuracy problems as the calculations surpass 11 significant digits.

5.4 Conclusion

This chapter has covered the Testing and Validation portion of the software life cycle. The Validation section compared the twelve functional areas of the system software with the system requirements. It was found that the software was indeed meeting the requirements, and therefore meeting the needs of the user. The Testing section looked at whether the subroutines were performing their desired functions correctly. Test results from ICECAP-PC routines were compared with answers from other CAD packages, from separate routines on the VAX and CYBER computers, and from hand calculations. Some routines were found to have operating limits, such as the root finder. Other routines were found to be extremely accurate. It is hoped that these test and validation results will form a baseline for improvements or modifications that might occur as part of a follow-on thesis effort.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

6.1 Introduction

This chapter presents the conclusions reached during this investigation. It also gives recommendations for those interested in continuing the development of ICECAP-PC.

The most significant result of this development is that a large complex software system can be designed and implemented on the common personal computer (64K RAM and two disk drives). Using structured programming and software engineering techniques, a complex system can be divided into smaller modules performing specific functions. These modules are functionally independent, easy to comprehend, and can be combined to form larger and more complex functions. A very complex software system, such as a CAD package for control system design and analysis, can and was constructed from these small, functionally independent modules.

6.2 Conclusions

A computer-aided design program for control system design and analysis (ICECAP-PC) was created. The development of ICECAP-PC followed standard software engineering practices (9). And followed the classic software life-cycle development sequence: Planning Phase, Requirements Definition Phase, Preliminary Design Phase, Detailed Design Phase, Implementation Phase, Testing Phase, and the Operations and Maintanence Phase (10).

A description of the requirements of the entire design package was

presented in Chapter 2. These requirements were consolidated from sources identifed during the literature search that was accomplished during the Planning Phase. Once these requirements were defined, the software development effort then had a direction to proceed.

The next goal was to select the resources and organize the system requirements into a functional structure. This was accomplished in the Preliminary Design Document, which is composed of Chapter 3 (Preliminary Design), Appendix C (system structure charts), and Appendix D (system data dictionary).

During the combined Detailed Design and Implementation Phase the functional structure was implemented in a specific language on a particular machine. This aspect of the process is documented in Chapter 4 (Implemented Design) and Appendix E (system source code).

During the entire software development effort, both subroutine level and system level testing was performed. Results of these tests are included in Chapter 5 (Testing and Validation).

Overall, software engineering techniques were used successfully to decompose a complex system into easily manageable parts. These techniques greatly simplified the development of the ICECAP-PC software and permitted the thesis effort to meet its objectives.

The prototype microcomputer software development workbench (MICROSDW) environment was modified to work on two new operating systems, MS-DOS and PC-DOS. It was also modified to work on two different families of personal computers, the Z-100 family and the IBM-PC family. This modified MICROSDW, ICECAP-PC, was then used to create a menu-driven shell that acts as an interface between the user and the control system analysis modules.

In order to enhance the capabilities of the selected design language, TURBO-Pascal, a library of scientific and engineering functions were created, e.g. arctangent and common logorithms (log base ten).

The detailed design was completed on all of the priority one modules. All of the data input routines were fully implemented as well as all of the major ICECAP-PC utilities. This allows the ICECAP-PC package to accept and decode user commands, accept the user input (transfer functions, polynomials, matrices, etc.), provide on-line assistance, provide error detection/protection, and perform the desired mathmatical manipulations. The available manipulation routines include polynomial and matrix manipulation routines, time response, frequency response, partial fraction expansion, and system specifications.

Due to time constraints, this thesis, the appendicies, and the currently available software diskettes do not include the frequency and time response modules. These modules are available as a separate document. The incorporation of these modules requires extensive memory management (e.g. overlays).

6.3 Recommendations

The first and most important recommendation is that the design and implementation od ICECAP-PC should be continued and expanded. The development of the sub-routines as presented in the Requirements Definition Chapter should be continued, with emphasis on completing the priority one modules. A proposed order of development of the remaining priority one modules is presented below:

- Root-locus analysis
  - -- Tabular output of the root-locus
  - -- Plot the root-locus on the terminal
  - -- Calculate the root-locus for a branch of interest, zeta of interest, or gain of interest
- Graphical output for time response, frequency response, and root-locus analysis

The addition of these capabilities will provide a basic control system design and analysis tool (7). To further enhance the capabilities of ICECAP-PC, the priority two modules should be developed. The priority two modules include:

- Laplace domain analysis
- Discrete control analysis
- Modern control theory
- Block diagram manipulation

The eventual combination of priority one and priority two modules will provide the user with a sophisticated analysis tool. It is hoped that ICECAP-PC will become a tool that is useful not only in the classroom, but also out in the field of actual control system design and analysis.

Appendix A: ICECAP-PC User's Manual

This appendix contains a user's guide for the ICECAP-PC system. This is considered a 'living document' which reflects the current implementation of ICECAP-PC.

1. Introduction

This is the User's Manual for the ICECAP-PC Computer Aided Design (CAD) System. The ICECAP-PC program provides an environment for control system design and analysis hosted on a microcomputer. The system is menu driven and will provide on-line help upon request. The following is a list of the ICECAP-PC main menu options:

CHANGE - Allows the user to change analysis plane and sampling time.

COPY - Allows the user to copy one transfer function into another or one matrix into another.

DEFINE - Allows the user to define a transfer function, polynomial or a matrix.

DISPLAY - Displays the results, transfer functions, or matrices on the screen.

- Any submenu option with an (*) has not been implemented yet.

FORM - Forms a CLTF from an OLTF or a combination of GTF and HTF.

- Forms an OLTF from a combination of GTF and HTF.

MODIFY - Add a root or delete a root from a polynomial or transfer function.

- Change a single location of a matrix.

PRINT - Sends screen output to the printer.

- Any submenu option with and (*) has not been implemented yet.

RECOVER - Used to continue a previous session. Will copy files into ICECAP-PC memory.

STOP - Normal command for leaving ICECAP-PC.

SWITCHES - Flip control switches ON and OFF.

UPDATE - Allows user to save current session to a user specified file.

At the time of this publication the system consists of an installation program, BUILDDAT, and a user interface program, ICECAP-PC. BUILDDAT will configure the program ICECAP-PC for a host microcomputer. If you wish to change the configuration of ICECAP-PC please turn to the ICECAP-PC Programmer's Manual for guidance. ICECAP-PC provides the user with an interface to the control system design and analysis routines.

The User's Manual is divided into four major sections, Section II contains general operator notes and information, Section III describes how to use BUILDDAT, Section IV describes how to use ICECAP-PC, and Section V describes several common problems and their solution.

2. General Information

This is a living document. It is designed to reflect the CAD program ICECAP-PC as it is currently implemented. If any changes are made to the software that affects the options available this document should be changed to reflect those changes.

It is assumed by the author's that the users of this CAD package are familiar with the operation of a computer, and understand the basics of control system design and analysis.

2.1 Who Should Read This Section

This section should be read by all first time users of ICECAP-PC. This section provides general information about the operating systems and hardware required to operate this CAD package. As well as general information about ICECAP-PC.

2.2 Operating Systems

ICECAP-PC is designed to operate with the following disk operating systems:

- CP/M-80 version 2.2 or greater
- CP/M-86 version 1.1 or greater
- MS-DOS version 1.0 or greater
- PC-DOS version 1.0 or greater

If you are using MS or PC-DOS the following files must be on the disk that you boot the system with, CONFIG.SYS and ANSI.SYS The CONFIG.SYS file may not be on your DOS distribution disk, if not then you must create the file. The file only contains three lines:

```
DEVICE = ANSI.SYS
BREAK = ON
FILES = 16
```

This file may be created with a wordprocessor or the editor provided with your DOS. If the ANSI.SYS file is not on your DOS working disk, then copy this file from the original distribution disk.

2.3 Hardware Required

ICECAP-PC is designed to operate on a hardware system that contains a minimum of 64K RAM and a disk drive. Improved performance would be noted if a hard disk or a disk emulator were used instead.

There is an additional hardware requirement of an 80 column by 24 line monochrome monitor. The monitor is means by which ICECAP-PC communicates with the user.

Instructions in Section 3 and Section 4 are designed to be general. However, this manual provides separate instructions for a two floppy disk system and a hard disk system where applicable.

3. BUILDDAT

3.1 Who Should Read This Section

If the disks that you have for ICECAP-PC contain the executable code for your particular machine and the HELP.SYS and MICROSDW.SYS files, skip this section.

3.2 How to Use BUILDDAT

BUILDDAT reads five text files describing the hard-

ware environment (TERM.TXT, PRINT.TXT), help text (HELP.TXT), and the menu structure (MENU.TXT, PARAM.TXT). These text files must be present on the same disk as the BUILDDAT.COM file for BUILDDAT to access them and execute correctly. At a minimum the following files must be on your 'BUILDDAT' disk:

- BUILDDAT.COM
- TERM.TXT
- PRINT.TXT
- HELP.TXT
- MENU.TXT
- PARAM.TXT

BUILDDAT creates two system files, MICROSDW.SYS and HELP.SYS, and optionally a file containing a listing of what BUILDDAT did (BUILDDAT.OUT). The prompts the user receives are "yes" or "no" type questions, the default answer if the user simply enters a carriage return is, NO. The user may enter "yes" or "no" in response to the questions or merely enter "y" or "n". These entries may be made in either upper or lower case.

The user has the option to either recreate the entire MICROSDW.SYS and HELP.SYS files from the text files or to recreate the portion of the system files matching one of the text files. The user is first asked if he wants to rebuild the entire system file or selected portions of it. The user is then asked if he wants to spool the output from BUILDDAT to the file BUILDDAT.OUT.

If the user is rebuilding the entire system he will see information on the screen indicating which file is being processed. The next prompt the user will receive will ask whether he wishes to see/display the MENU structure. The next prompt the user will receive will ask whether he wishes to see/display the WORD structure.

If the user selects to rebuild portions of the system files he will then be prompted for which portions are to be rebuilt. The prompts appear in the following order:

- parameters	PARAM.TXT
- terminal	TERM.TXT
- printer	PRINT.TXT
- help messages	HELP.TXT
- menu structure	MENU.TXT

If the user elects to rebuild a portion of the system file

BUILDDAT will rebuild the portion selected and then prompt for the next section to be rebuilt.

After the text file(s) have been processed by BUILDDAT the MICROSDW.SYS file is rewritten with the information updated from the text files. The HELP.SYS file is rewritten immediately after the HELP.TXT file is processed.

If you choose to spool the information to the file BUILDDAT.OUT the menu structure and the word structure will not be displayed on the screen. That information will be written to the BUILDDAT.OUT file.

**** Floppy Disk Systems Only ****

Copy the HELP.SYS file and the MICROSDW.SYS file onto the same disk as the compiled version of ICECAP-PC.

WARNING: If you choose to recreate the entire system and spool the output to BUILDDAT.OUT only those files listed above should be on the disk. If there are more files on the disk there will be insufficent disk space to run BUILDDAT.

**** Hard Disk System Only ****

Insure the HELP.SYS and the MICROSDW.SYS files are in the same directory as the compiled version of ICECAP-PC.

4. ICECAP-PC

ICECAP-PC uses a hierarchical keyword command structure to prompt the user for inputs. These command words are arranged in a hierarchy of menus.

To execute the ICECAP-PC the user should first type:

"ICECAP-PC" (without quotes)

in response to the operating system prompt. ICECAP-PC will then display the message:

"Initializing the MICROSDW Menu structure
and Hardware configuration"

ICECAP-PC takes about 1 minute to initialize. ICECAP-PC will

then display the first page of the "title slide", indicating that initialization is complete. The user is then prompted to enter a "carriage return" to continue. The second page of the "title slide" will then be displayed and the user will once again be prompted to enter a carriage return to continue.

A list of keywords is then displayed on the screen followed by the prompt:

"Enter Option >"

see Figure 1. The user may then enter one of the keywords displayed or an abbreviation for a keyword. The minimum abbreviation for each keyword is shown in bold. The user may either enter the entire keyword or an abbreviation for the keyword with at least as many characters as the minimum abbreviation. All characters entered for a keyword are validated against a list of currently acceptable keywords.

CHANGE	**DISPLAY**	**MODIFY**	**RECOVER**	**SWITCHES**
COPY	**FORM**	**PRINT**	**STOP**	**UPDATE**
DEFINE	**HELP**			

Enter Option >

Figure 1

The user may enter more than one complete keyword or abbreviation after the "Enter Option >" prompt. Keywords and abbreviations must be separated by spaces and terminated with a carriage return. This permits the experienced user to type ahead without waiting for lower level menus to be displayed. After a keyword or sequence of keywords is entered ICECAP-PC checks them against the list of valid keywords. If the user entered an invalid keyword or abbreviation, the invalid keyword is highlighted and the user is prompted to reenter a valid keyword. After a valid keyword has been entered ICECAP-PC displays the keywords for the next lower level menu or executes the selected option. When then keywords are displayed for a lower level menu the previously entered keywords are displayed following "Enter Option >".

If a mistake is noted prior to pressing the carriage return <CR>, just use the backspace or delete key to erase backwards to the error. Correct the error and retype the remainder of the command. If the program is prompting you for a command completion, you can only erase the characters internally back to the point that began that prompt.

The user may "abort" a particular option while being prompted from the menu structure by entering a "$" and a "carriage return". This returns the user back to the top level menu.

4.1 Available Commands

The following is a detailed description of each of the available ICECAP-PC options.

4.1.1 CHANGE Command

The CHANGE command is used to initialize the CHANGE functions of ICECAP-PC. This function allows the user to select the plane of analysis and to change the sampling time, TSAMP. The proper format for the CHANGE command is:

```
Enter Option > CHANGE PLANE

Enter Option > CHANGE TSAMP
```

The system will provide a menu of allowable plane changes in response to the first command. This options has not yet been implemented.

4.1.2 COPY Command

The command COPY is used to move the contents of one object, called the source, to a second object called the destination. The objects may be transfer functions, matrices, or polynomials. The COPY command does not destroy the contents of the source. The proper format of the COPY command is as follows:

```
Enter Option> COPY <source_name> <destination_name> <CR>
```

For example:

```
Enter Option> COPY GTF OLTF <CR>
                or
Enter Option> COPY MATA MATB <CR>
```

ICECAP-PC will supply a menu of allowable transfer functions, matrices, and polynomials for both the source and the destination. ICECAP-PC will only allow you to copy a transfer function into a transfer function, a matrix into a matrix,

or a polynomial into a polynomial.

4.1.3 DEFINE Command

The command DEFINE is used to initialize the transfer functions, polynomials and matrices of ICECAP-PC so that further calculations using those functions and/or matrices may be executed. The input to the system is set with this command.

4.1.3.1 Transfer Function/Polynomial Input

If you try to manipulate a variable that has not been defined the system will prompt you for the necessary information. ICECAP-PC will provide a menu of objects that may be defined. If you select a transfer function or a polynomial the system will then prompt you to select either factored or polynomial input formats. The proper fomat for the command is:

```
Enter Option> DEFINE (transfer function) (fact/poly)

Enter Option> DEFINE (polynomial) (fact/poly)
```

The system will prompt you for the required data. If you provide incorrect data for the prompt ICECAP-PC will describe the error and prompt you for input again. For example, let us assume you have entered the command:

```
Enter Option> DEFINE GTF POLY
```

The system prompt will be:

```
What is the degree of the numerator... ( max of 10 ) ?
```

You in turn should enter the integer value of the numerator degree. ICECAP-PC will then display the following prompt:

```
... the denominator ( max of 10 ) ?
```

Again input an integer value. ICECAP-PC will then draw a display screen for the transfer function. ICECAP will highlight the first input area and provide a prompt:

```
Your number ...
```

Input the real number you would like in the input area. If

you enter a valid real number, the value is stored, and the highlighted area moves to the next input area. This continues until all of the input areas are filled. If you have selected the factored input format, and have entered the first half of a complex conjugate pair, ICECAP-PC will enter the second half of the pair. If you try to enter the first half of a complex conjugate pair in the last root location, ICECAP-PC will tell you this is an invalid input and prompt you to enter the root again.

If you selected the polynomial input format and the leading coefficient of the polynomial was anything other than zero, the leading coefficent is set to one and all succeeding coefficients are divided by the leading coefficient. The constant/gain is then multiplied by the leading coefficient. ICECAP-PC will then compute the roots of the polynomial. The transfer function or polynomial is then stored. So the user may verify the defined object, it is displayed to the user.

NOTE: If you input a transfer function or a polynomial with a non-unity leading coefficient the displayed object will not be the same as the one you entered, it will, however, be the algebraic equivalent.

If you selected the factored input format, ICECAP-PC will compute the polynomial, store the object, then display the object from storage.

4.1.3.2 Matrix Definition

The matrix definition option of ICECAP-PC will also prompt you for any required information. The proper format of this command is:

Enter Option> DEFINE (matrix)

ICECAP-PC will then ask:

How many rows in your matrix (max of 10) ?

In response enter an integer, once you have correctly entered an integer ICECAP-PC will then ask:

...columns (max of 10) ?

You are limited to a max of 10 rows and a max of 10 columns. ICECAP-PC will draw the display screen for the matrix, high-

light the input area and provide the prompt:

Your number...

Input the real number you would like in the input area. If you enter a valid real number, the value is stored, and the highlighted area moves to the next input area. This continues until all of the input areas are filled.

After the user has fully defined the matrix it is stored. The matrix is then displayed to the user so that he may verify it is the way he desires before it is used in manipulations.

4.1.4 DISPLAY Command

The command DISPLAY is used to write information onto the terminal screen. This information generally takes the form of plots of system response, the listing of the system specification, or a root locus plot. DISPLAY may also be used to display the current contents of transfer functions and matrices in the ICECAP-PC data base. The proper format of the DISPLAY command is:

Enter Option > DISPLAY (object)

ICECAP-PC will provide a menu of allowable 'objects' in response to the DISPLAY command. Any 'object' followed by an asterisk (*) has not been implemented yet.

4.1.4.1 DISPLAY OLTF

This option will DISPLAY the open-loop transfer function on the terminal.

4.1.4.2 DISPLAY CLTF

This option will DISPLAY the closed-loop transfer function on the terminal.

4.1.4.3 DISPLAY GTF

This option will DISPLAY the feed forward transfer function on the terminal.

4.1.4.4 DISPLAY HTF

This option will DISPLAY the feed back transfer function on the terminal.

4.1.4.5 GAIN

This option has not been implemented yet.

4.1.4.6 BUTRWRTH

This option has not been implemented yet.

4.1.4.7 BESSEL

This option has not been implemented yet.

4.1.4.8 EQUATION

This option has not been implemented yet.

4.1.4.9 FREQ/RESP

This option has not been implemented yet.

4.1.4.10 LOCUS

This option has not been implemented yet.

4.1.4.11 LOC/GAIN

This option has not been implemented yet.

4.1.4.12 LOC/BRAN

This option has not been implemented yet.

4.1.4.13 MATRIX

If you select this option you are presented with a

lower level menu. All of the available options are described below.

The ADD, SUBTRACT, and MATXMULT will add, subtract, or multiply two matrices together and store the result in a user selected location. You will be prompted for the three matrix names.

The INVERSE command will invert a matrix and store the matrix in the user selected location. If you desire both the inverted matrix and the storage matrix may be the same.

The TRANSPOSE command will tranpose a matrix and store the matrix in a user selcted location. If you desire both the transposed matrix and the storage matrix may be the same.

The SCLRMULT command will multiply a matrix by a scalar and store the result in a user specified location. If you desire both the original and the resulting matrices may be the same.

The matrix name commands will display the named matrix to the user.

4.1.4.14 MODERN

This option has not been implemented yet.

4.1.4.15 NICHOLS

This option has not been implemented yet.

4.1.4.16 NYQUIST

This option has not been implemented yet.

4.1.4.17 INYQUIST

This option has not been implemented yet.

4.1.4.18 PAR/FRAC

This option has not been implemented yet.

4.1.4.19 POLY

If you select this option you are presented with a lower level menu. All of the available options are described below.

The ADD, SUBTRACT, and POLYMLT options will prompt you for the names of the two polynomials you wish to ADD, SUBTRACT, or MULTIPLY together. ICECAP-PC will also prompt you for the name of the polynomial that you wish the new polynomial stored into. If you desire you may add POLYA and POLYA together and store the result in POLYA.

If you input one of the thirteen polynomial names the polynomial of your choice will be displayed on the screen.

4.1.4.20 RICATTI

This option has not been implemented yet.

4.1.4.21 ROUTH

This option has not been implemented yet.

4.1.4.22 SPECS

This option has not been implemented yet.

4.1.4.23 SWITCHES

This option has not been implemented yet.

4.1.4.24 TIME/RESP

This option has not been implemented yet.

4.1.5 FORM Command

The FORM command is used to produce the CLTF from either the OLTF or a combination of GTF and HTF. The formulas for producing the CLTF is as follows:

$$CLTF = (GAIN * GTF) / (1 + GAIN * GTF * HTF)$$

CLTF = (GAIN * OLTF) / (1 + GAIN * OLTF)
CLTF = GTF + HTF (In Parallel)

The FORM command will also form the OLTF from the GTF and the HTF. The formula for that operation is:

OLTF = GTF * HTF

4.1.6 HELP Command

The HELP command will present the user with a two option submenu. If you select the SYSTEM option you will be presented with an overall system description. It contains a short description of all the main menu options. If you select the FUNCTION option you will be presented with a submenu that looks like the main menu. When you select one of the options, a detailed description of the options will be presented to the user.

4.1.7 MODIFY Command

The modify command is used to change polynomials and matrices without redefining the entire polynomial or matrix. If you desire to change a transfer function this may be done by modifying the numerator polynomial, the denominator polynomial or both. After selecting the MODIFY option the user is presented with three choices, ADDROOT, DELROOT, and CHANGE.

4.1.7.1 ADDROOT Command

The ADDROOT command is used to modify or insert a root into a polynomial. In response to the ADDROOT command ICECAP-PC will provide a menu of available polynomials. An example of the command line is:

Enter Option> ADDROOT ODPOLY

After you have correctly entered the name of the polynomial ICECAP-PC will display the polynomial and prompt you with:

Your number ...

In addition the real area of the additional root will be highlighted. In response to the prompt enter a real number. You will then be prompted for the imaginary part of the

root. If any input other than zero is made ICECAP-PC will calculate the conjugate of this root. After you have correctly input the root ICECAP-PC will recompute the polynomial and it will be displayed on the screen.

4.1.7.2 DELROOT Command

The DELROOT command is used to remove or delete a root from a a polynomial. In response to the DELROOT command ICECAP-PC will provide a menu of available polynomials. An example of a command line is:

Enter Option> DELROOT ONPOLY

After you have correctly enetered the name of a polynomial ICECAP-PC will display the polynomial and prompt your with:

The number of the root you wish to delete is...

The numbers of the roots can be found to the left of the factored display. If the root you have chosen to delete is complex the conjugate will also be deleted. After ICECAP-PC has recomputed the new polynomial it will be displayed on the screen.

4.1.7.3 CHANGE Command

The CHANGE command is used to modify a single location in a matrix. In response to to CHANGE command ICECAP-PC will provide a menu of matrices that may be modified. An example of the command line is:

Enter Option > CHANGE MATA

After you have correctly entered the name of the matrix ICECAP-PC will display the matrix and prompt you with:

The row of the location you wish to modify ?

Enter a valid row number. The system will then prompt:

...the column ?

Enter a valid column number. ICECAP-PC will then highlight the location and provide the prompt:

Your number...

Enter a real number in response to the prompt. ICECAP-PC will then store the new matrix in the correct location and re-display it to you.

4.1.8 PRINT Command

The command PRINT is used to write information to an external file as well as to the terminal screen. The contents of the external file can be sent to a printer at the conclusion of the design session. PRINT is most often used to summarize the last iteration of a design session. If you wish to view the data before it is sent to the external file the DISPLAY command may be used. Further instructions for DISPLAY may be found using the HELP command. This is a good way to ensure a 'clean copy' to the printer. The proper format for the PRINT command is as follows:

```
Enter Option> PRINT (object)
```

ICECAP will provide a menu of allowable 'objects' in response to the PRINT command. Any 'object' followed by an asterisk (*) has not been implemented yet. This option is not currently available.

4.1.9 RECOVER Command

RECOVER is used to copy user specified files into ICECAP-PC memory so that you can continue a previous session at the point where you left off. The user files were previously specified with the UPDATE command. You may only RECOVER files that exist on the disk.

CAUTION: Be sure you input the transfer function and polynomial file name in response to the 'tf&pols.dat' prompt, and the matrix file name in response to the 'matrix.dat' prompt. If you mix the two ICECAP-PC will think there is NO data in either of its internal files.

4.1.10 STOP Command

The STOP command is used to exit gracefully from ICECAP-PC. Information is always stored in the ICECAP-PC 'tf&pols.dat' and the 'matrix.dat' files for later use. If you desire to save this information in a user specified file, use the UPDATE command. STOP is the normal mode for leaving

ICECAP-PC.

4.1.11 SWITCHES Command

The command SWITCHES is used to flip various control switches ON and OFF. This function has not yet been implemented.

4.1.12 UPDATE Command

The UPDATE command may be used to periodically write all of the contents of ICECAP-PC memory files, 'tf&pols.dat' and 'matrix.dat' to user specified files. ICECAP-PC will provide prompts for the user name for each of the files. This command is particularly useful when there is more than one user of ICECAP-PC or a single user is working on more than one design in parallel. In order to write the user specified files into the ICECAP-PC files use the RECOVER command.

The file name you specify should be eight characters or less, and should not contain any blank spaces. The file need not exist on the disk. If the files do exists on the disk ICECAP-PC will over-write them. If they don't exist ICECAP-PC will create them.

5. If Problems are Encountered

The following is a brief list of actions to take when problems are encountered during execution of ICECAP-PC.

5.1 Errors During Initialization

Make sure all files required by the ICECAP-PC reside on the same disk as ICECAP-PC. Attempt to re-execute the program. The files that must be on the disk are:

- ICECAPPC.COM
- ICECAPPC.000
- ICECAPPC.001
- ICECAPPC.002
- HELP.SYS
- MICROSDW.SYS
- TF&POLS.DAT
- MATRIX.DAT

5.2 A "trashy" Title Slide Appears

1. Attempt to re-execute the program.

2. If the error occurs again and this is the first attempt to execute ICECAP-PC on a new system or after installing a new terminal definition, the terminal definition needs to be corrected, refer to the Programmer's Manual for detailed instructions.

3. If the ICECAP-PC has previously been executed on the current system with no problems either the MICROSDW.SYS file has been damaged or the executable ICECAP-PC files has been damaged, restore the former files from a backup copy and try again.

Appendix B: Programmer's Guide

This appendix contains a programmer's guide for the ICECAP-PC system. This guide provides instructions and insight into the installation program, BUILDDAT, as well as the control system design and analysis package, ICECAP-PC.

ICECAP-PC Programmer's Manual

1. Introduction

This is the Programmer's Manual for the ICECAP-PC Computer Aided Design (CAD) System. The purpose of this manual is to supply information about the structure of ICECAP-PC, and to provide procedures for changing the configuration of the program.

The ICECAP-PC program provides an environment for control system design and analysis hosted on a microcomputer. At the time of this publication, the system consists of an installation program, BUILDDAT, and a user interface program, ICECAP-PC. BUILDDAT configures the program ICECAP-PC for a host microcomputer. ICECAP-PC provides the user with an interface to the control system design and analysis routines. This manual contains information on both BUILDDAT and ICECAP-PC.

The Programmer's Manual is divided into five major sections. Section 2 contains general information concerning both programs, section 3 contains information on the installation program BUILDDAT, section 4 covers ICECAP-PC, the user interface, and section 5 discusses the Development Diskettes.

Before beginning any type of development work on the ICECAP-PC program, it is essential that the programmer read this manual completely. Of particular importance is the information contained in section 2.

2. General Information

This section contains important information concerning both BUILDDAT and ICECAP-PC. Before attempting to compile executable versions of these programs, please read sections 2.1 and 2.2 thoroughly.

Section 2.1 covers operating system limits on the number of open files. When running BUILDDAT, these limits are reached, so a method for increasing the number of open files allowed by the operating system is presented.

Section 2.2 discusses why the IBM and Z-100 versions had to be different. There are certain files that must be present on the executable diskette for the IBM that are not required for the Z-100.

2.1 Limits on Open Files

In normal configuration, MS-DOS allows up to eight files to be open simultaneously when executing a program. (This is true for both Zenith MS-DOS version 2 and IBM MS-DOS version 2.10, which were used in the ICECAP-PC development effort.) The BUILDDAT program, however, requires more than eight open files when it executes. If BUILDDAT runs on a system where the default is eight files, TURBO Pascal will issue an I/O error message F3, "too many files open". (TURBO Pascal version 2.0 and 3.0 allow up to 16 open files, so TURBO is actually echoing an operating system error message.) The file limit must be increased for BUILDDAT to execute properly.

Increasing the open file limit involves modification of the CONFIG.SYS file on the operating system diskette. If this file does not already exist, it can be created using the operating system's line editor. Equivalently, the file can be created with a word processing program, as long as the program has a "non-document" mode. The CONFIG.SYS file must contain the following statement:

```
FILES=16
```

This statement tells the operating system to increase the number of open files allowed from 8 to 16. When the microcomputer system is booted, the operating system will automatically check the CONFIG.SYS file and implement any configuration specified there. Therefore, whenever executing BUILDDAT, the system must be booted with the modified

CONFIG.SYS file.

NOTE: As of this writing, only the BUILDDAT program needs the higher file limit. The ICECAP-PC program can still be executed with the default value of 8 open files.

2.2 IBM and Z-100 Differences

The ICECAP-PC program has two versions; one for the Zenith Z-100, and the other for the IBM PC/XT/AT. The differences are due to the distinct manner in which these systems implement cursor movement on the screen. Because of the extensive use of cursor control in ICECAP-PC, how each system implements this control becomes an important issue. When compared with the IBM, the Z-100 has a much simpler implementation of escape codes for accomplishing cursor movement. The IBM, on the other hand, requires additional files on the executable diskette and a special procedure to reroute output before escape codes will function properly. These additional requirements for the IBM are discussed in detail in sections 2.2.1 and 2.2.2.

2.2.1 Additional Files for the IBM

Cursor movement in ICECAP-PC is handled through the use of escape code sequences. For instance, clearing the screen and homing the cursor is accomplished by the sequence "ESC E" for the Z-100, and by "ESC [2 J" for the IBM (where ESC stands for escape). The Z-100 recognizes escape codes in its normal operating mode. The IBM, however, does not. To get the IBM to recognize escape code sequences, one must implement the "extended screen and keyboard control" feature.

The "extended screen and keyboard control" feature is covered in chapter 2 of the Disk Operating System Technical Reference Manual (version 2.10). This feature requires the creation of a CONFIG.SYS file on the DOS boot disk. The CONFIG.SYS file is the same file that was mentioned in section 2.1 of this Programmer's Manual. The file should contain the command

```
DEVICE=ANSI.SYS
```

where ANSI.SYS is an already existing DOS system file. This command causes DOS to replace the standard screen and keyboard drivers with the extended features. Escape code sequences will then be recognized by the operating system.

This affects implementation of the IBM version of ICE-CAP-PC in the following way. The executable code for ICE-CAP-PC, plus all the associated system, data, and overlay files, must be copied to a diskette containing the following files:

IBMBIO.COM
IBMDOS.COM
COMMAND.COM
ANSI.SYS
CONFIG.SYS

The first three files are operating system files and can be placed on the diskette by using the /S option in the FORMAT command. IBMBIO.COM and IBMDOS.COM are both hidden files and will not show up in the disk directory. These two files, along with COMMAND.COM, must be on the disk to be able to boot the system. The fourth file, ANSI.SYS, can be copied from the DOS diskette. It contains the information needed for the "extended screen and keyboard control" feature. The fifth file, CONFIG.SYS, was discussed earlier, and it contains the command DEVICE=ANSI.SYS. When the operating system boots, it checks the disk directory for the CONFIG.SYS file, then executes the commands in it. This will enable the "extended screen and keyboard control" feature, allowing the IBM to recognize escape codes. Therefore, to have the escape code sequences operate properly on the IBM, the ICECAP-PC program must run on a system booted with the five files mentioned above.

2.2.2 Additional Procedures for the IBM

Because TURBO Pascal was chosen as the language for ICECAP-PC, some additional procedures had to be added to the main program to get escape code sequences to work properly on the IBM. These additional procedures are the reason why ICECAP-PC has separate Z-100 and IBM versions.

During the development of ICECAP-PC, it was found that the "extended screen and keyboard control" feature, as explained in the previous section, did not provide cursor control on the IBM, even though the proper steps were followed. The problem was caused by TURBO Pascal. Specifically, how TURBO sends output to the screen was the trouble.

Normally, in a Pascal compiler other than TURBO, such as Microsoft's Pascal Compiler 2.00, output from WRITE and WRITELN statements are routed through the operating system

(MS-DOS) and then out to the screen (the standard output device). The "extended screen and keyboard control" feature requires that the output be routed this way, i.e. through the operating system. This means that escape code sequences would work properly if Microsoft's Pascal Compiler was used. However, problems surfaced because TURBO Pascal was the chosen language.

In contrast to a normal Pascal compiler, TURBO Pascal routes WRITE and WRITELN statements through the Basic Input/ Output System (BIOS). This increases the execution speed of TURBO by bypassing the operating system, but it also prevents the escape code sequences from functioning. Since the "extended screen and keyboard control" feature only works with the operating system, a procedure had to be developed to route the output from WRITE and WRITELN statements back through MS-DOS instead of BIOS.

The procedure was called standard_output and was placed in the file STDOUT.PAS. This procedure is needed only for the IBM. Its main function is to use DOS function call number 2, which sends output to the standard output device, i.e. the display. (For more information on DOS function calls, see chapter 5 in the DOS Technical Reference Manual version 2.10, by Microsoft Corp.) The code for procedure standard_output is listed below.

```
procedure standard_output(c: char);
  var
    r   :   regpak;
  begin
    r.ah := 2;
    r.dl := ord(c);
    msdos(r);
  end;
```

Notice that the variable "r" is declared to be of type "regpak". Regpak is a record containing all the registers needed to pass values back and forth to MS-DOS. The type declaration for regpak is added to the program ICECAPPC in the file ICECAPPC.PAS. This is what makes the IBM version of ICECAPPC.PAS different from the Z-100 version. The type definition for regpak is shown below.

```
regpak  =  record
     al,ah,bl,bh,cl,ch,dl,dh  :  byte;
     ax,bx,cx,dx,bp,si,di,ds,es,flags  :  integer;
   end;
```

There are two other lines of code in the IBM version of ICECAPPC.PAS that are not included in the Z-100 version. The first is an "include" statement to include the file STDOUT.PAS.

```
{$I stdout.pas}
```

This is the file containing the procedure standard_output. The second is a statement which changes the address of ConOutPtr.

```
ConOutPtr := Ofs(standard_output);
```

ConOutPtr is a pointer used by WRITE and WRITELN statements in TURBO Pascal to locate the portion of code which outputs data to the terminal. By placing the address of the new procedure standard_output into ConOutPtr, then output normally directed by TURBO Pascal through BIOS now goes through the procedure standard_output. Standard_output then redirects the output to MS-DOS. From now on, all output to the terminal via WRITE or WRITELN will be redirected through ANSI.SYS, thereby allowing the IBM PC to recognize escape codes.

Because of the distinct manner in which the IBM PC implements escape codes, ICECAP-PC had to be produced in an IBM and a Z-100 version. The differences are confined to only two files, ICECAPPC.PAS and STDOUT.PAS. STDOUT.PAS is required for the IBM version only.

3. BUILDDAT - The Installation Program

The menu system of ICECAP-PC is produced by the program BUILDDAT. BUILDDAT uses text files which contain information about the terminal and printer of a microcomputer system. The program uses these files to install the menu on that particular system. Other text files containing data on the menu and help systems are also processed by BUILDDAT. The product of the BUILDDAT program is a user-defined menu structure which is installed on a user-defined microcomputer system.

3.1 Compiling BUILDDAT from Source Files

There are 22 source files which must be compiled to form an executable version of BUILDDAT. These files are written in TURBO Pascal and are identified with a .PAS extension. (TURBO Pascal is produced and copyrighted by Borland International Inc.) All 22 files are listed below.

BUILDDAT.PAS	ERRORMSG.PAS
ADDKEY.PAS	GETWORD.PAS
ADDTOMEN.PAS	LOCMENU.PAS
ADDWORD.PAS	MAKEMENU.PAS
AVL-1.PAS	MAKEPROC.PAS
AVL-2.PAS	MENUPRNT.PAS
AVL-3.PAS	READMENU.PAS
AVL-4.PAS	MENUCONS.PAS
DEFCALL.PAS	MENUTYPE.PAS
DEFMENU.PAS	MSDWCONS.PAS
DISPROC.PAS	MSDWTYPE.PAS

Only the BUILDDAT.PAS file needs to be compiled. The other 21 files are "include" files, and will automatically be compiled with BUILDDAT.PAS. Use the "com-file" option in the TURBO Pascal compiler (TURBO Pascal version 2.0 or 3.0 can be used for compiling). This produces the executable file BUILDDAT.COM and saves it on the disk. Once the file BUILDDAT.COM exists, it can be executed by typing BUILDDAT.

3.2 Running BUILDDAT

When BUILDDAT.COM executes, it reads the five text files listed below and converts them into data arrays. Text files are identified by a .TXT extension.

HELP.TXT

MENU.TXT
PARAM.TXT
PRINT.TXT
TERM.TXT

As BUILDDAT.COM runs, it creates two system files:

MICROSDW.SYS
HELP.SYS

And it also creates an optional file, listing what BUILDDAT.COM did:

BUILDDAT.OUT

The HELP.SYS file contains the system help messages, while the MICROSDW.SYS file contains the entire menu structure for ICECAP-PC. These two .SYS files must be present on the same disk as the executable ICECAPPC.COM file for the ICECAP-PC program to operate successfully.

The five .TXT files are extremely important because they define the hardware parameters, as well as the help and menu system structure. They are now discussed individually. (NOTE: For further information on running BUILDDAT, see the ICECAP-PC User's Manual.)

3.2.1 The HELP.TXT File

HELP.TXT contains the help messages and any other type of message displayed to the user as a prompt or as feedback. The file is an ASCII text file. Messages can be multiple lines, with up to 79 characters per line. Each message is separated by a <$> symbol on a separate line. The entire HELP.TXT file ends with a <$$> symbol on a separate line. The number of messages allowed in the HELP.TXT file is determined by the constant num_msg_dir in the MSDWCONS.PAS file. To increase the number of messages, num_msg_dir must also be increased. The number of lines of message text available in the file HELP.TXT is determined by the constant num_msg_line, also in the file MSDWCONS.PAS. It is possible that by increasing the number of messages, the number of available lines might be exceeded. This problem is solved by increasing the value of num_msg_line.

During the development of ICECAP-PC, it was found that memory could be saved by placing as many messages, prompts, and displays, as possible, into the HELP.TXT file. This

prevents a lot of pure text information from using up precious memory space in the ICECAPPC.COM file or in the overlay files. The HELP.TXT file is converted into the HELP.SYS file by BUILDDAT. The HELP.SYS file can almost be considered "unlimited" in space when compared to the 64K limit on the ICECAPPC.COM file. By keeping most of the text information in HELP.SYS, memory space can be freed for more important functions.

3.2.2 The MENU.TXT File

The words used in the ICECAP-PC menu system are stored in the file MENU.TXT. When BUILDDAT executes, the procedure make_menu creates the static data structure from MENU.TXT and stores it in the system file MICROSDW.SYS. This system file describes the hardware environment, as well as the menu structure, for ICECAP-PC. The MENU.TXT file allows the specification of a hierarchical menu structure. The system is designed so that a sub-menu may have more than one parent menu. Common sub-menus reduce the amount of space required to store the menu structure.

Processing of the MENU.TXT file by BUILDDAT is controlled by two constants in the MSDWCONS.PAS file; num_ptrs and num_words. Num_ptrs determines the number of pointers available in the menu data structure. Num_words places a limit on the number of different words in the menu. When executing ICECAPPC.COM, if some words are present on the menu, but others are missing, then the value of num_words is probably too low. If, however, the title slide displays correctly, but all the menu words are just random characters, then the value of num_ptrs is too low.

There are two areas to become familiar with before attempting to construct a MENU.TXT file. One is the grammar of the file, and the other is its structure. These areas are discussed individually in the next two sections.

3.2.2.1 MENU.TXT File Grammar

There are six different types of input text lines that may be found in the MENU.TXT file: menu definition, call definition, menu word, comment, end of menu definition, and end of menu. The grammar for the MENU.TXT file is defined as follows:

$ = menu definition

. = call definition
; = comment, when placed at the beginning of a line
! = end of menu definition
: = end of MENU.TXT file

There are two different types of menu definitions. The first allows the specification of a menu option followed, on separate lines, by the menu words of the sub-menu. This type of specification is ended with a '!', signifying the end of the menu definition. An example is shown below.

```
$KEYWORD1
OPTIONA
OPTIONB
!
```

The second type of menu definition allows the assignment of a previously defined sub-menu to be the sub-menu of a new menu option. For example, if KEYWORD2 and KEYWORD3 are both new menu options which have exactly the same sub-menu as KEYWORD1 in the above example, then the menu definitions can be written as shown below.

```
$KEYWORD2 = KEYWORD1
$KEYWORD3 = KEYWORD1
```

Call definitions specify the procedure or routine to be called by a particular menu option. A period '.' at the beginning of a line indicates a call definition. However, periods between two menu words indicate sub-menu options. For instance, if OPTIONA calls procedurea and OPTIONB calls procedureb, the call definitions would be written as shown below.

```
.KEYWORD1.OPTIONA = procedurea
.KEYWORD1.OPTIONB = procedureb
```

Comments may be placed on any line in the MENU.TXT file as long as a semi-colon ';' is the first character on the line. The end of the MENU.TXT file is denoted by a colon ':' on the very last line.

Menu words are words that will show up in the actual menu. In the above examples, KEYWORD1, KEYWORD2, KEYWORD3, OPTIONA, and OPTIONB are all menu words.

3.2.2.2 MENU.TXT File Structure

The structure of the MENU.TXT file begins with the definition of the root menu. Since all the menu words in the root menu are on the same level, the '$' symbol is not needed. Each menu word in the root menu is listed on a separate line. The end of the list is denoted by a '!'.

Following the root menu, each of the root menu words are then further divided into either sub-menus or procedure call definitions. To define the options under the root menu, use the following sequence:

```
$<root menu word>
<first sub-menu word>
<second sub-menu word>
        .
        .
        .
<last sub-menu word>
!
```

Each of the sub-menu words may in turn have sub-sub-menus, and so on. The following sequence is used to specify a lower level menu:

```
$<root menu word>.<menu word>.<menu word>.etc.
<first sub-sub-menu word>
<second sub-sub-menu word>
        .
        .
        .
<last sub-sub-menu word>
!
```

The menu definition line, i.e. the one starting with the '$', must specify the complete path using menu words.

To write a call definition for a root menu word, the following sequence is used:

```
.<root menu word> = <routine>
```

For a lower level menu, call definitions look like the following:

```
.<root menu word>.<menu word>.<menu word>.etc = <routine>
```

There are two important points to remember when using call definitions. First, every menu path must end in a call definition. And second, in the call definition line, there

must be a blank space before and after the equal sign '='.

A sample MENU.TXT file is shown below.

```
; define the root menu options
KEYWORD1
KEYWORD2
KEYWORD3
!
; define the sub-menu under KEYWORD1
$KEYWORD1
OPTIONA
OPTIONB
!
; define the routines called by OPTIONA
; and OPTIONB
.KEYWORD1.OPTIONA = procedurea
.KEYWORD1.OPTIONB = procedureb
; KEYWORD2 and KEYWORD3 have exactly the
; same structure as KEYWORD1, so they can
; be defined by one line each.
$KEYWORD2 = KEYWORD1
$KEYWORD3 = KEYWORD1
:
```

3.2.3 The PARAM.TXT File

The PARAM.TXT file contains parameters for initializing output devices, such as a terminal or printer, and also has parameters for enabling transaction or temporary files, where output can be stored.

The file is divided into four groups of ten parameters each. For the ICECAP-PC program, only the first group is needed. The other three parameter groups are ignored, but must be present in the file for BUILDDAT to execute properly. (For more information on the other parameter groups, see Appendix G in Vincent Parisi's MS thesis).

In the first group, lines 1 through 8 are used by ICECAP-PC. Lines 9 and 10 are not used at all. For each line, a 0 represents an enabled device, and a 1 means the device is disabled. As of this writing, ICECAP-PC has only three enabled devices; crt output, terminal input, and display status line.

3.2.4 The PRINT.TXT File

Since printers have not yet been implemented in ICECAP PC, the PRINT.TXT file contains only dummy data. The file is fifty lines long. Each line has a line number, followed by a space, followed by the data, and terminated with a carriage return.

3.2.5 The TERM.TXT File

The TERM.TXT file contains escape code sequences and graphics characters for a particular terminal. Versions of TERM.TXT exist for the Zenith Z-100 and the IBM monochrome screens. Previous thesis efforts produced TERM.TXT files for the Heathkit H-19/29 terminal and the DEC Rainbow 100. During the development of ICECAP-PC, it was found that the TERM.TXT file for the IBM monochrome screen also worked for the Amdek amber terminal and the Princeton Graphics Enhanced Color Display.

The TERM.TXT file is 95 lines long. Each line contains a line number, followed by a space, followed by the data, and terminated with a carriage return. An optional comment can be added between the data and the carriage return, as long as the comment is preceeded by a space. The entire file is divided into areas for each escape sequence. (For a complete description of each area, see Appendix G of Vincent Parisi's MS thesis).

As an example, lines 27 through 33 are reserved for the escape code sequence for entering reverse video. For the Z-100, this sequence is "ESC p". The first line of the reserved area, line 27, contains the number of characters in the escape sequence; in this case, 2. The next two lines contain the decimal equivalent of the ASCII characters in the escape code. In this example, "ESC p" becomes "27 112". The remaining lines are filled with zeroes. For the IBM monochrome screen, the escape sequence for reverse video is "ESC [7 m". In the TERM.TXT file, line 27 would now contain a 4, because the escape sequence is 4 characters long. The next four lines would contain "27 91 55 109", the decimal equivalent of the ASCII characters "ESC [7 m". Modifications to the TERM.TXT file can be accomplished with the operating system's line editor, or with a word processor that has a non-document mode. This shows how easily the TERM.TXT file can be modified for any type of terminal, as long as the escape code sequences are known.

4. ICECAP-PC - The User Interface

The program ICECAP-PC provides the user with a menu-driven interface to control system design and analysis tools. The program is built in a manner which makes it easy to add routines and expand capabilities.

4.1 Compiling ICECAP-PC from Source Files

There are 42 source files which must be compiled to form an executable version of ICECAP-PC. These files are written in TURBO Pascal and are identified with a .PAS extension. All 42 files are listed below.

*ICECAPPC.PAS	MODIFY.PAS
BODE.PAS	MSDWCONS.PAS
CONCONS.PAS	MSDWTYPE.PAS
COPY.PAS	MSG.PAS
DEFINE.PAS	OUTPUT.PAS
DELROOT.PAS	PAUSE.PAS
DISP.PAS	POLY.PAS
DISPLAYC.PAS	POLYMAN.PAS
FORM.PAS	PROCESER.PAS
GETCOM.PAS	PROMPTCM.PAS
*GETDAT.PAS	PROMPTHE.PAS
GETINT.PAS	READCOM.PAS
GETLINE.PAS	REALS.PAS
GETMAT.PAS	RECOVER.PAS
GETSTRIN.PAS	SELECT.PAS
GETTF.PAS	*STDOUT.PAS
HELP.PAS	TERMINAL.PAS
INROOT.PAS	TRIM.PAS
INSTRUCT.PAS	UCASE.PAS
MATRIX.PAS	UPDATE.PAS
MATRXMAN.PAS	VALNDEC.PAS

The files that are preceeded by asterisks are different, depending on which microcomputer system is used. ICECAPPC.PAS has a Zenith Z-100 version and an IBM-PC version. If interchanged, these versions will not work on the other system. GETDAT.PAS also has a Z-100 version and an IBM-PC version. However, the only difference between the versions is one line of code which displays the name of the computer on the title slide. If interchanged, GETDAT.PAS will not prevent operation of the program. STDOUT.PAS is an IBM unique file. The IBM version of ICECAPPC.PAS will automatically include the STDOUT.PAS file when compiling. The Z-100 version of ICECAPPC.PAS does not include this file because it is not

needed. If STDOUT.PAS is present on the Z-100 disk by mistake, it will not affect operation of the program because it is ignored during compilation.

To compile the 42 source files, only the ICECAPPC.PAS file needs to be compiled. The other 41 files are "include" files, and will automatically be compiled with ICECAPPC.PAS. Use the "com-file" option in the TURBO Pascal compiler. This produces the executable file ICECAPPC.COM and saves it on the disk.

4.2 The Use of Overlays in ICECAP-PC

Due to the number of source files in ICECAP-PC, a compiler limit was reached in TURBO Pascal. TURBO allows a maximum size of 64K for a .COM file. By compiling all 42 source files, this limit is exceeded. To overcome this limit, overlay procedures are used.

Overlays allow the creation of programs which are much larger than the computer's memory. Overlay procedures are stored in one or more files separate from the main program. When an overlay procedure is called, it is taken from the overlay file where it resides, and automatically loaded into the overlay area reserved in the main program. It is essential that a procedure in one overlay file not call another procedure in the same overlay file, because both overlay procedures cannot exist in the same reserved area at the same time. However, it is permissible to have overlay procedures call each other, if they exist in two different overlay files. This is because overlay files have different reserved areas in main memory.

To create an overlay, the reserved word 'overlay' must be added to the declaration of a procedure or function (e.g. overlay procedure filename, or overlay function filename). The following procedures in ICECAP-PC are declared as overlays:

recover
update
get_location
move_tf
move_poly
move_matrix
inroot
delroot
chgmat

When compiling the 42 source files with the above overlay procedures, an executable ICECAPPC.COM file is produced, along with two overlay files as shown below.

ICECAPPC.COM
ICECAPPC.000
ICECAPPC.001

Overlay files are always identified by a numbered extension. The order in which the source files are included during compilation determines which procedures end up in each overlay file. The chart below lists each overlay procedure, the source file in which it originated, and the overlay file where it resides.

overlay procedure	source file	overlay file
recover	RECOVER.PAS	000
update	UPDATE.PAS	000
get_location	COPY.PAS	000
move_tf	COPY.PAS	000
move_poly	COPY.PAS	000
move_matrix	COPY.PAS	000
inroot	INROOT.PAS	001
delroot	DELROOT.PAS	001
chgmat	MODIFY.PAS	001

Through the use of overlays, ICECAP-PC can be expanded far greater than its present form. However, while main program memory is saved, the use of overlays requires disk access during execution. Careful consideration, therefore, is essential when creating overlays, to prevent excessive time spent on disk operations.

5. The Development Diskettes

There are five development diskettes for ICECAP-PC. These disks contain the source and executable code for both BUILDDAT and ICECAPPC. The code on these disks is documented in appendix E and F of this thesis, and reflects the status of ICECAP-PC at the time this thesis was written. Documentation is also kept on the DEC VAX 11/780 computer in the Information Sciences Laboratory (room 245). The files are in Dr. Lamont's account, under the directory [LAMONT.ICECAPPC].

Development diskettes exist in both a Zenith Z-100 and an IBM PC/XT/AT version. The labels on the diskettes identify which version applies.

The five development diskettes contain the following information:

Disk 1	BUILDDAT source files
Disk 2	BUILDDAT text and executable files
Disk 3	ICECAP-PC source files
Disk 4	more ICECAP-PC source files
Disk 5	ICECAP-PC executable files

Each of the diskettes are discussed further in the next five sections.

5.1 Disk 1 - BUILDDAT Source Files

Disk 1 contains the 22 source files which must be compiled to form an executable version of BUILDDAT. The procedure for compiling BUILDDAT is covered in section 3.1 of this manual. For completeness, a list of the 22 source files is presented below.

BUILDDAT.PAS	ERRORMSG.PAS
ADDKEY.PAS	GETWORD.PAS
ADDTOMEN.PAS	LOCMENU.PAS
ADDWORD.PAS	MAKEMENU.PAS
AVL-1.PAS	MAKEPROC.PAS
AVL-2.PAS	MENUCONS.PAS
AVL-3.PAS	MENUPRNT.PAS
AVL-4.PAS	MENUTYPE.PAS
DEFCALL.PAS	MSDWCONS.PAS
DEFMENU.PAS	MSDWTYPE.PAS
DISPROC.PAS	READMENU.PAS

5.2 Disk 2 - BUILDDAT Text and Executable Files

Disk 2 contains the executable file BUILDDAT.COM, the five text files that BUILDDAT uses as it executes, the two system files that BUILDDAT creates, and the optional file BUILDDAT.OUT. Procedures for running BUILDDAT are covered in the ICECAP-PC User's Manual.

Because of BUILDDAT's requirement for 16 open files (as addressed in section 2.1 of this manual), the operating system files and the appropriate CONFIG.SYS file are included on Disk 2. This means the microcomputer system can be booted from Disk 2, so BUILDDAT will execute properly. A list of the files is shown below.

operating system files (hidden)
COMMAND.COM
CONFIG.SYS
BUILDDAT.COM
HELP.TXT
MENU.TXT
PARAM.TXT
PRINT.TXT
TERM.TXT
HELP.SYS
MICROSDW.SYS
BUILDDAT.OUT

5.3 Disk 3 - ICECAP-PC Source Files

Due to the large number of ICECAP-PC source files, they had to be divided between two disks. Disk 3 contains all the include files in ICECAPPC.PAS (floppy disk version) that do not have a disk identifier. Most of these files are related to previous thesis work accomplished by Vincent Parisi and Paul Moore. Of the 42 ICECAP-PC source files, 23 reside on Disk 3. They are listed below.

*ICECAPPC.PAS	OUTPUT.PAS
CONCONS.PAS	PAUSE.PAS
DISPLAYC.PAS	PROCESER.PAS
GETCOM.PAS	PROMPTCM.PAS
*GETDAT.PAS	PROMPTHE.PAS
GETINT.PAS	READCOM.PAS
GETLINE.PAS	*STDOUT.PAS
GETSTRIN.PAS	TERMINAL.PAS
INSTRUCT.PAS	TRIM.PAS
MSDWCONS.PAS	UCASE.PAS

MSDWTYPE.PAS
MSG.PAS
VALNDEC.PAS

The files ICECAPPC.PAS and GETDAT.PAS have an IBM and a Z-100 version. The IBM version of each of these files exists only on the IBM development disk, while the Z-100 version exists only on its corresponding disk. STDOUT.PAS applies only to the IBM, so it is included only on the IBM disk.

5.4 Disk 4 - More ICECAP-PC Source Files

Disk 4 contains all the include files in ICECAPPC.PAS (floppy disk version) that have the disk identifier "b:". These files are all new code generated during the ICECAP-PC development effort. There are 19 source files on this disk, as listed below.

BODE.PAS
COPY.PAS
DEFINE.PAS
DELROOT.PAS
DISP.PAS
FORM.PAS
GETMAT.PAS
GETTF.PAS
HELP.PAS
INROOT.PAS
MATRIX.PAS
MATRXMAN.PAS
MODIFY.PAS
POLY.PAS
POLYMAN.PAS
REALS.PAS
RECOVER.PAS
SELECT.PAS
UPDATE.PAS

5.5 Disk 5 - ICECAP-PC Executable Files

Disk 5 contains all the files necessary for an operational version of ICECAP-PC. It includes the executable file plus any overlays, the system files, and the data files. The entire list of files is shown below.

ICECAPPC.COM
ICECAPPC.000
ICECAPPC.001
HELP.SYS
MICROSDW.SYS
MATRIX.DAT
TF&POLS.DAT

When ICECAPPC executes, it creates other data and transaction files such as MACRO.INP, PRINTER.OUT, TEMP.OUT, LIST_DEV, and BODE.DAT. These files do not have to be present to run the program, because ICECAPPC will create them

as needed. Therefore, they are not included on Disk 5.

The IBM PC/XT/AT version has some additional files on Disk 5. These are the files discussed in section 2.2.1 of this manual. The additional files are required so the IBM will recognize escape code sequences. They include the operating system files and the configuration files, as listed below.

IBMBIO.COM (hidden)
IBMDOS.COM (hidden)
COMMAND.COM
ANSI.SYS
CONFIG.SYS

For the escape codes to work correctly in the IBM version of ICECAP-PC, the microcomputer system must be booted from a disk containing the above files.

Appendix C: System Structure Charts

Introduction

This appendix presents the implementation structure and hierarchical charts. The charts relate directly the system design as implemented. These charts are designed to be read from the top down. The first time a node is encountered the passed parameters are specified, thereafter, an asterisk (*) is used to indicate the parameters were previously specified. If a node does not show either specified parameters or an asterisk, there are no parameters passed to or from the node. Two different cross references are presented below.

Node to Procedure

This is a node to procedure cross reference for the Appendix C Structure Charts. Any procedure that does not call any other procedure is marked by an asterisk (*).

NODE	PROCEDURE NAME
0	icecappc
1	get_cmd
1.1	gotoxy *
1.2	readcom
1.2.1	get_strng *
1.2.2	ucase *
1.3	get_line *
1.4	val_n_dec
1.4.1	check_word
1.4.2	trim *
1.5	prompt_help
1.5.1	out_string *
1.5.2	SVideoLow *
1.5.3	SVideoBold *
1.6	displa_commandword
1.7	prompt_cmd
1.7.1	highlight *
1.7.2	nohighlight *
1.8	instruction
1.9	proces_error
1.9.1	disp_msg
1.9.1.1	disp_line
1.9.1.2	clear_msg
1.10	clear

```
2                       pause
3                       ClearScreen *
4                       get_data
4.1                     title_slide
4.1.1                   rectangle
4.1.1.1                 graphics *
4.1.1.2                 nographics *
4.2                     bld_stat_line *
5                       select_routine
5.1                     define
5.1.1                   gettf
5.1.1.1                 get_int
5.1.1.2                 poly
5.1.1.2.1               get_unfact
5.1.1.2.1.1             make_pretty
5.1.1.2.1.1.1           out_int *
5.1.1.2.1.2             get_r_num
5.1.1.2.1.2.1           get_real
5.1.1.2.1.3             out_real *
5.1.1.2.2               roots
5.1.1.2.3               get_fact
5.1.1.2.4               form_poly *
5.1.1.3                 disptf
5.1.2                   getmat
5.1.2.1                 make_pretty_small_matrix
5.1.2.1.1               left_bracket
5.1.2.1.2               right_bracket
5.1.2.2                 make_pretty_large_matrix_one
5.1.2.3                 get_matrx_entries
5.1.2.3.1               make_pretty_large_matrix_two
5.1.3                   get_poly
5.1.3.1                 disppoly
5.2                     help
5.3                     disp
5.3.1                   mmatrix
5.3.1.1                 disp_matrx
5.3.1.2                 get_matrx_name
5.3.1.3                 matrx_manipl
5.3.1.3.1               matrxadd
5.3.1.3.2               mmatrxmlt
5.3.1.3.3               matrxsub
5.3.1.4                 matrx_manip2
5.3.1.4.1               matrxtran *
5.3.1.4.2               matrxinv
5.3.1.4.3               smatrxmlt *
5.3.2                   ppoly
5.3.2.1                 get_poly_name
5.3.2.2                 polmanip
5.3.2.2.1               polymlt
```

5.3.2.2.2	polyadd
5.3.2.2.3	polysub
5.3.2.3	polmanip2
5.3.2.3.1	spolymlt
5.4	ccopyy
5.4.1	get_location *
5.4.2	move_poly *
5.4.3	move_tf *
5.4.4	move_matrix *
5.5	modify
5.5.1	inroot
5.5.2	delroot
5.5.3	chgmat
5.6	recover
5.7	update
5.8	form
5.8.1	poly_from_storage
5.8.2	poly_into_storage

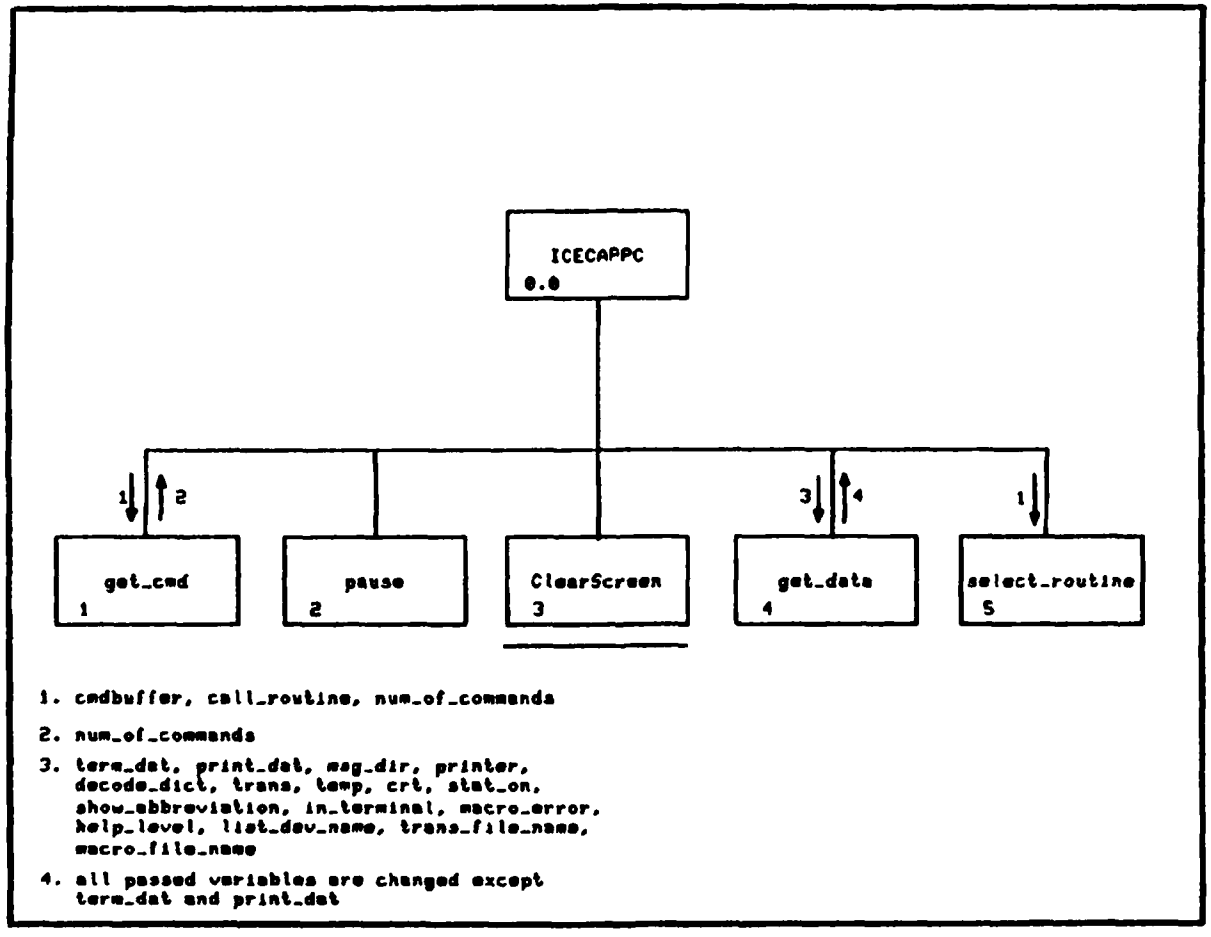
Procedure to Node

This is a procedure to node cross reference for the Appendix C Structure Charts. Any procedure that does not call any other procedure is marked by an asterisk (*).

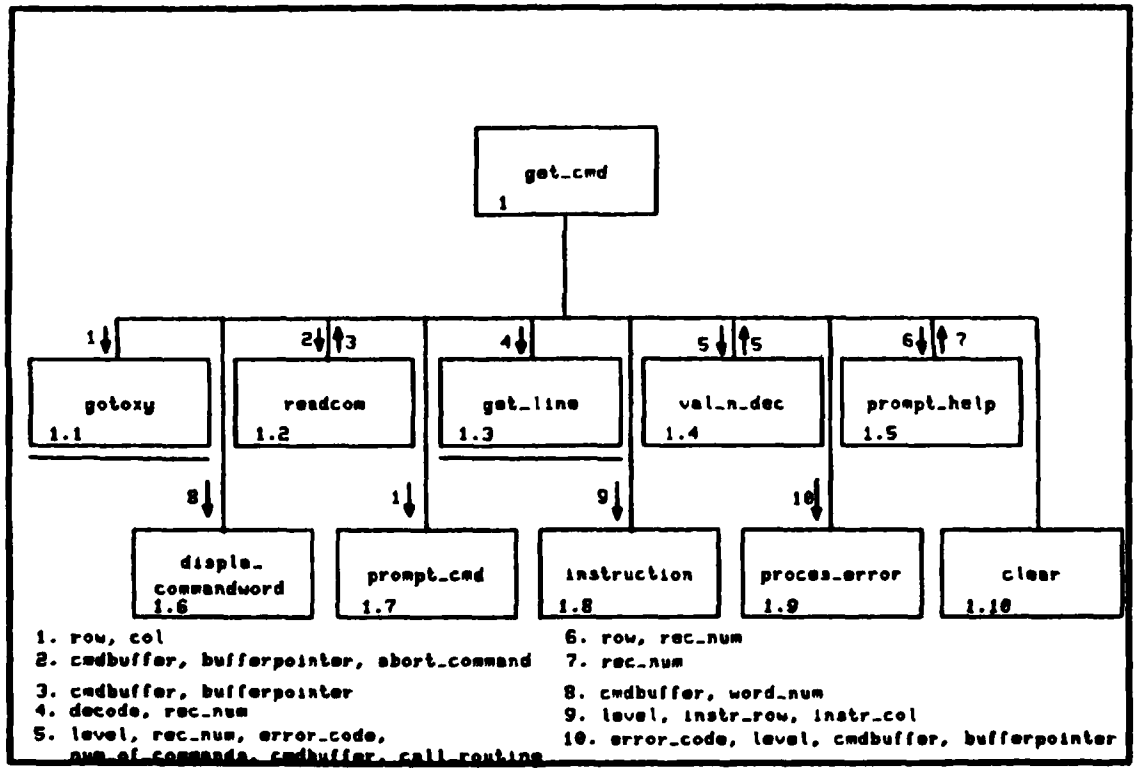
PROCEDURE NAME	NODE
bld_stat_line *	4.2
ccopyy	5.4
check_word	1.4.1
chgmat	5.5.3
ck_chr	
clear	1.10
clear_msg	1.9.1.2
ClearScreen *	3
define	5.1
del_lst_chr	
delroot	5.5.2
disp	5.3
displa_commandword	1.6
disp_line	1.9.1.1
disp_matrx	5.3.1.1
disp_msg	1.9.1
disppoly	5.1.3.1
disptf	5.1.1.3
form	5.8
form_poly *	5.1.1.2.4
getchi	
get_cmd	1
get_data	4
get_fact	5.1.1.2.3
get_int	5.1.1.1
get_line *	1.3
get_location *	5.4.1
getmat	5.1.2
get_matrx_entries	5.1.2.3
get_matrx_name	5.3.1.2
get_poly	5.1.3
get_poly_name	5.3.2.1
get_real	5.1.1.2.1.2.1
get_r_num	5.1.1.2.1.2
get_strng *	1.2.1
gettf	5.1.1
get_unfact	5.1.1.2.1
gotoxy *	1.1
graphics *	4.1.1.1
help	5.2

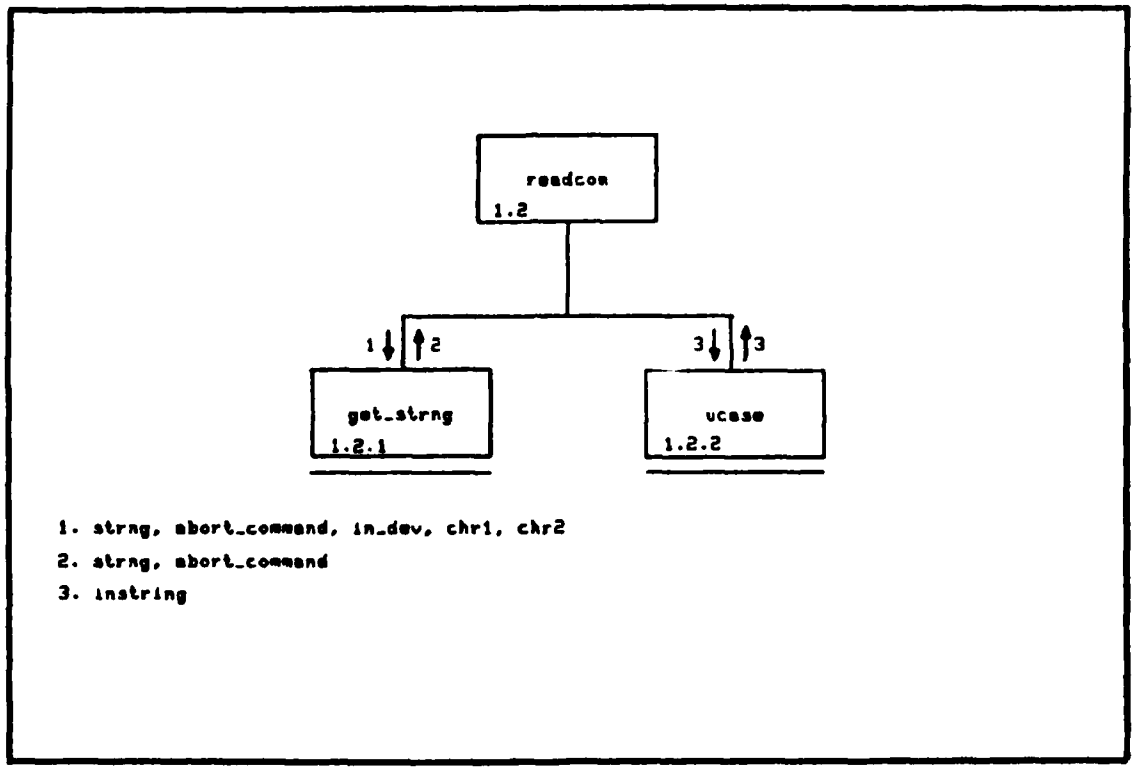
highlight *	1.7.1
icecappc	0
inroot	5.5.1
instruction	1.8
left_bracket	5.1.2.1.1
make_pretty	5.1.1.2.1.1
make_pretty_large_matrix_one	5.1.2.2
make_pretty_large_matrix_two	5.1.2.3.1
make_pretty_small_matrix	5.1.2.1
matrxadd	5.3.1.3.1
matrxinv	5.3.1.4.2
matrx_manip1	5.3.1.3
matrx_manip2	5.3.1.4
matrxsub	5.3.1.3.3
matrxtran *	5.3.1.4.1
mmatrix	5.3.1
mmatrxmlt	5.3.1.3.2
modify	5.5
move_matrix *	5.4.4
move_poly *	5.4.2
move_tf *	5.4.3
nographics *	4.1.1.2
nohighlight *	1.7.2
out_int *	5.1.1.2.1.1.1
out_real *	5.1.1.2.1.3
out_string *	1.5.1
pause	2
polmanip	5.3.2.2
polmanip2	5.3.2.3
poly	5.1.1.2
polyadd	5.3.2.2.2
poly_from_storage	5.8.1
poly_into_storage	5.8.2
polymlt	5.3.2.2.1
polysub	5.3.2.2.3
ppoly	5.3.2
proces_error	1.9
prompt_cmd	1.7
prompt_help	1.5
readcom	1.2
recover	5.6
rectangle	4.1.1
right_bracket	5.1.2.1.2
roots	5.1.1.2.2
select_routine	5
smatrxmlt *	5.3.1.4.3
spolymlt	5.3.2.3.1
SVideoBold *	1.5.3
SVideoLow *	1.5.2

title_slide	4.1
trim *	1.4.2
ucase *	1.2.2
update	5.7
val_n_dec	1.4
videobold	
videolow	

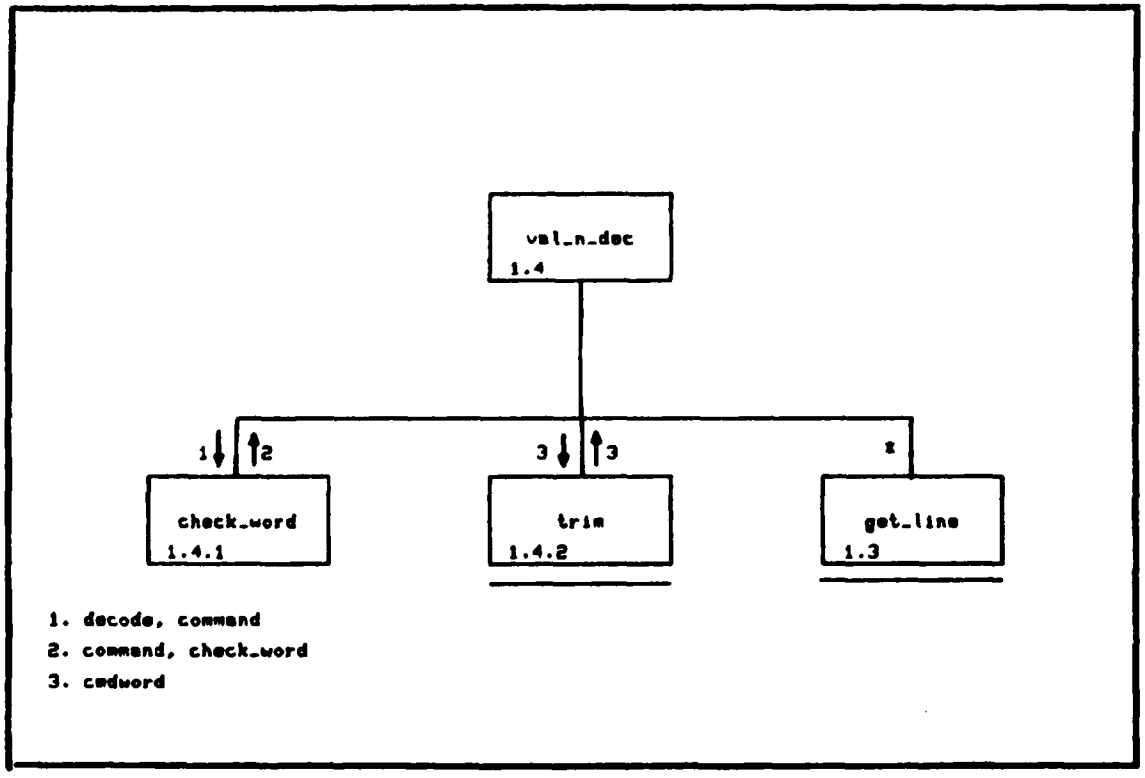


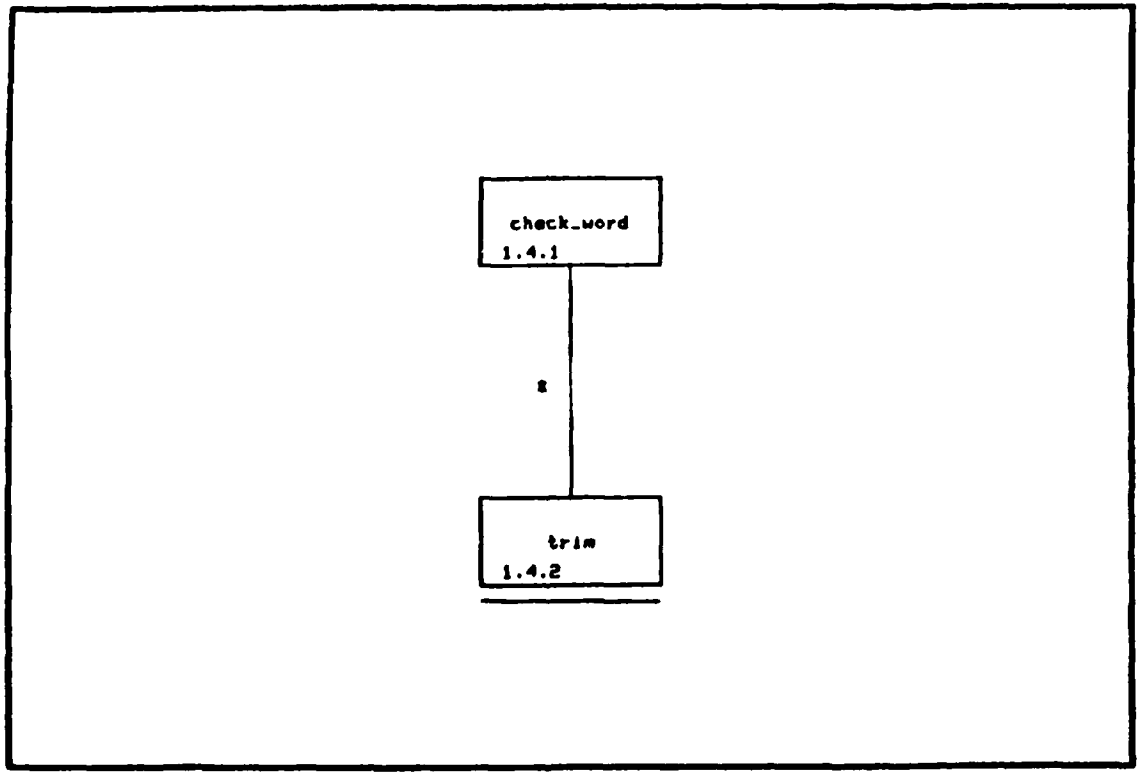
1. cmdbuffer, call_routine, num_of_commands
2. num_of_commands
3. term_dat, print_dat, msg_dir, printer, decode_dict, trans, temp, crt, stat_on, show_abbreviation, in_terminal, macro_error, help_level, list_dev_name, trans_file_name, macro_file_name
4. all passed variables are changed except term_dat and print_dat

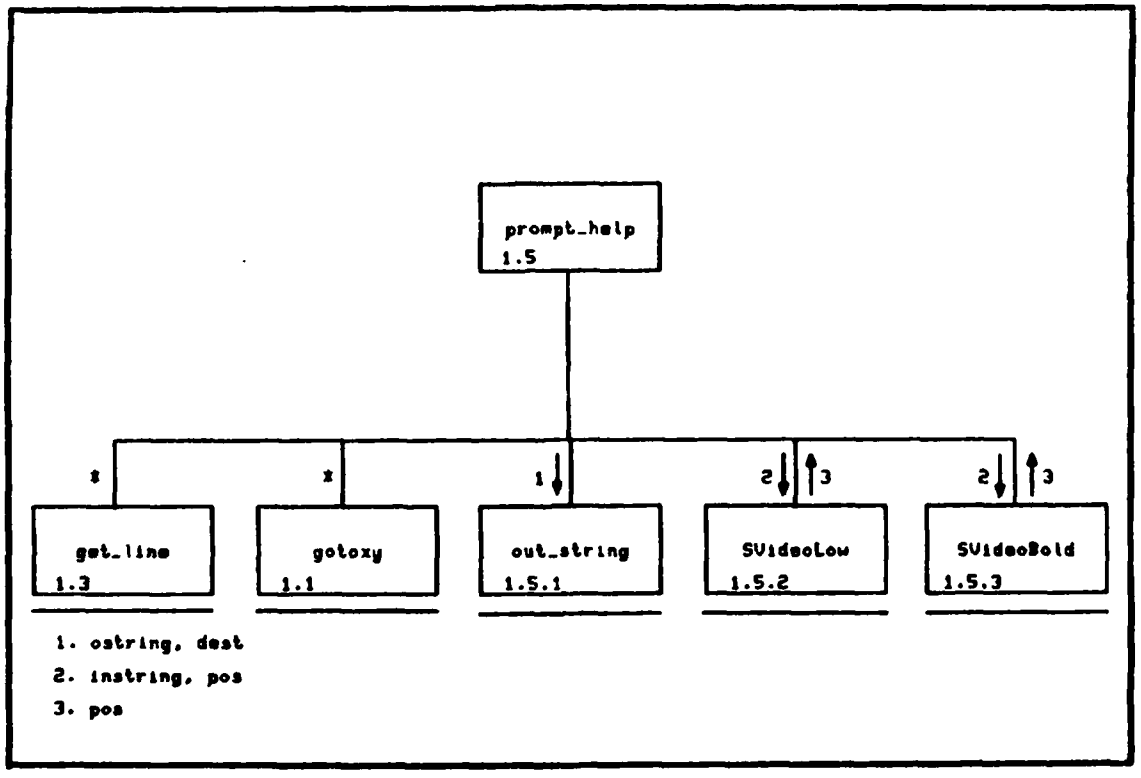

get_cmd
1
1
2
3
4
5
5
6
7
gotoxy
1.1
readcom
1.2
get_line
1.3
val_n_dec
1.4
prompt_help
1.5
8
1
9
10
displa_
commandword
1.6
prompt_cmd
1.7
instruction
1.8
proces_error
1.9
clear
1.10
1. row, col
2. cmdbuffer, bufferpointer, abort_command
3. cmdbuffer, bufferpointer
4. decode, rec_num
5. level, rec_num, error_code,
num_of_commands, cmdbuffer, call_routine
6. row, rec_num
7. rec_num
8. cmdbuffer, word_num
9. level, instr_row, instr_col
10. error_code, level, cmdbuffer, bufferpointer

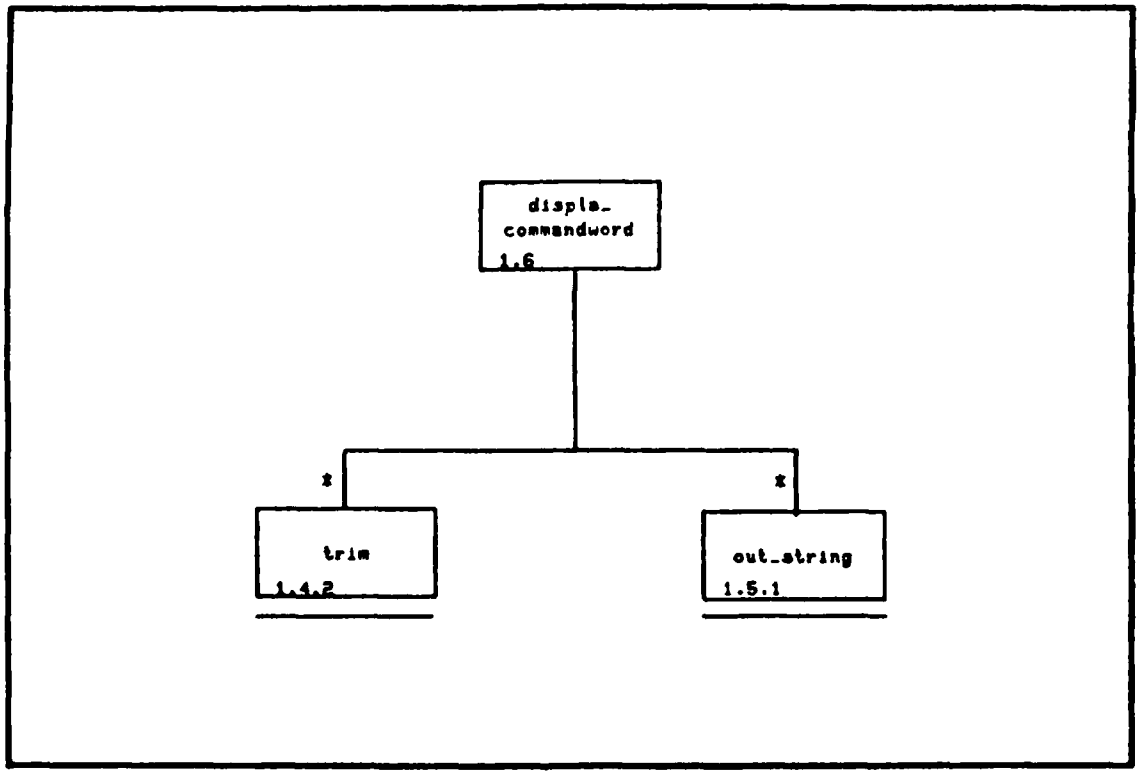


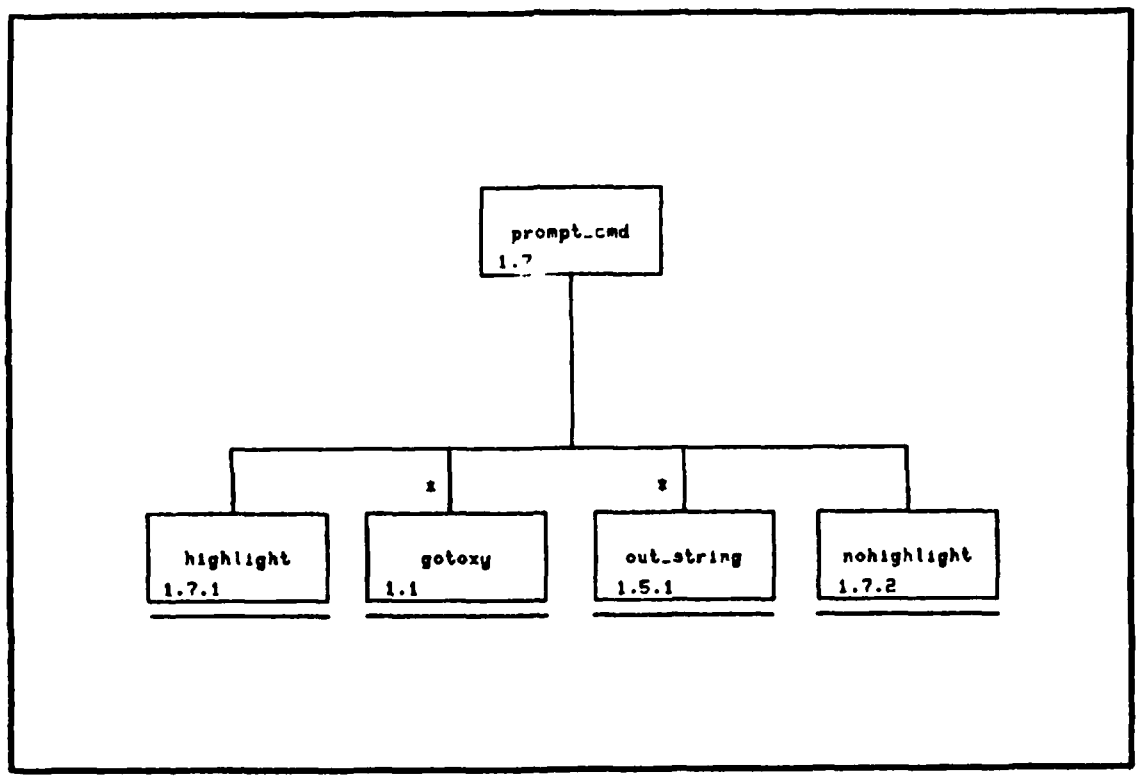
1. strng, abort_command, in_dev, chr1, chr2
2. strng, abort_command
3. instring

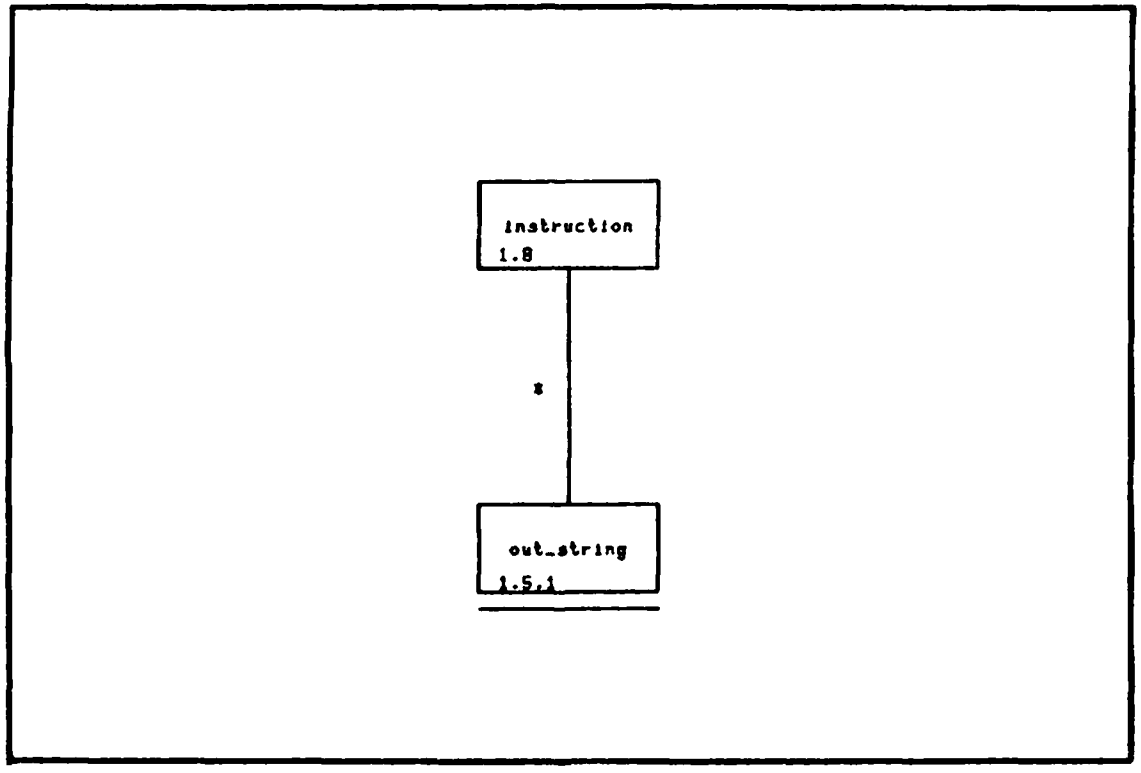

val_n_dec
1.4
1
2
3
3
x
check_word
1.4.1
trim
1.4.2
get_line
1.3
1. decode, command
2. command, check_word
3. cmdword

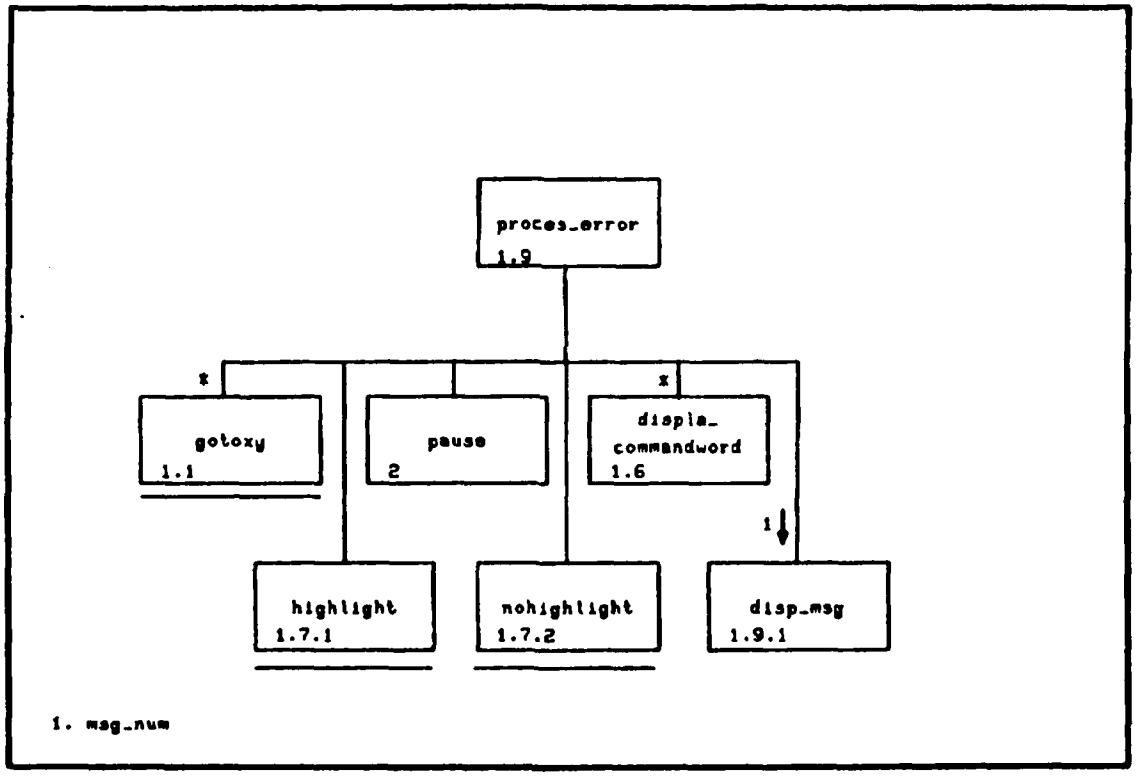

check_word
1.4.1
1
trim
1.4.2

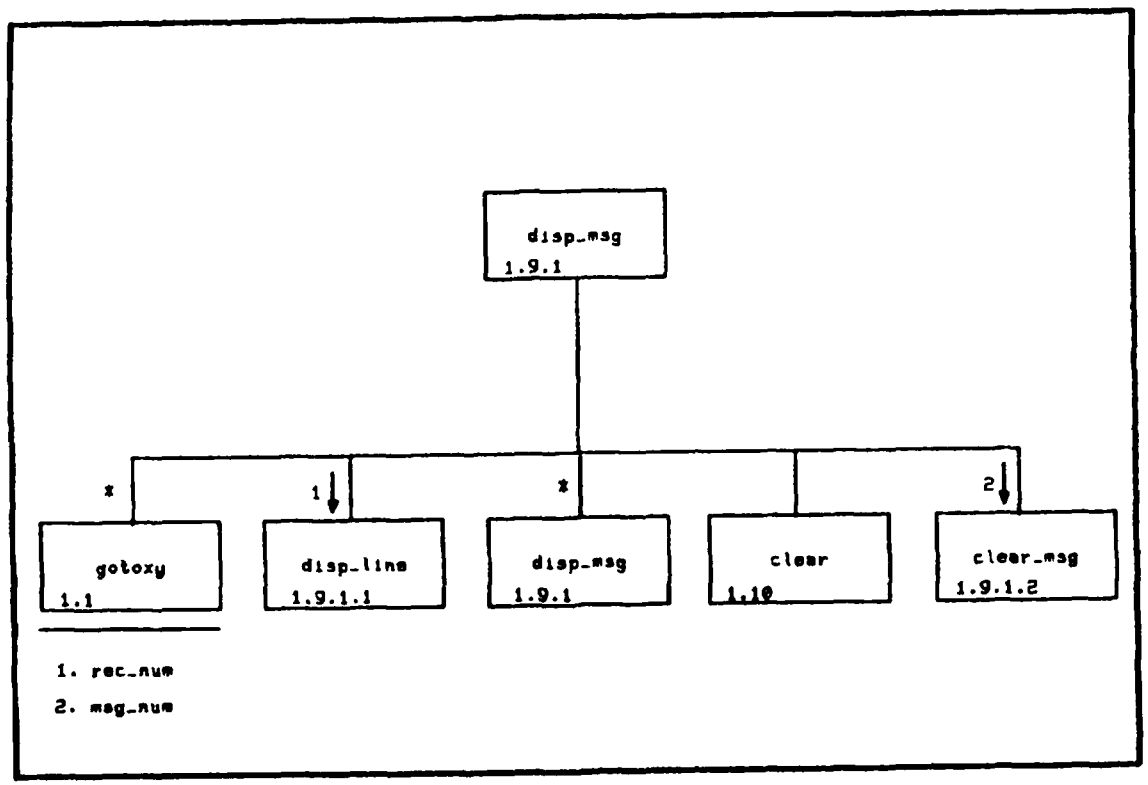

prompt_help
1.5
x
x
1
2
3
2
3
get_line
1.3
gotoxy
1.1
out_string
1.5.1
SVideoLow
1.5.2
SVideoBold
1.5.3
1. ostring, dest
2. instring, pos
3. pos

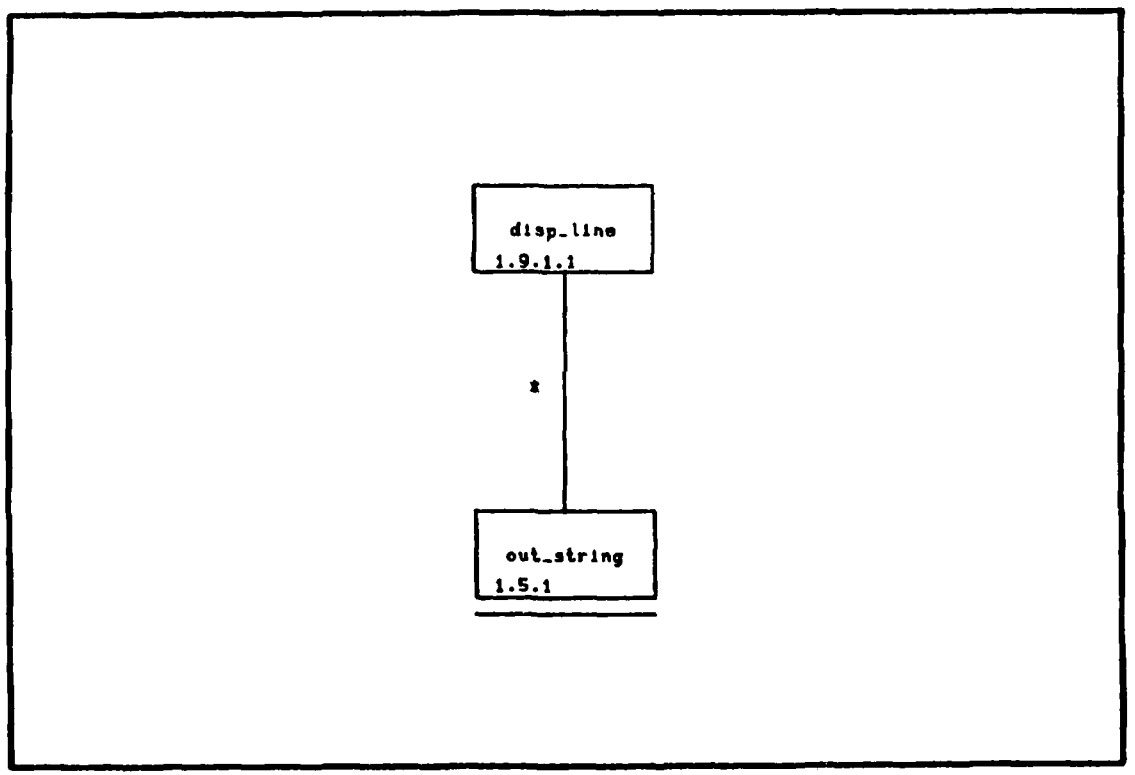

displa_
commandword
1.6
x
x
trim
1.4.2
out_string
1.5.1

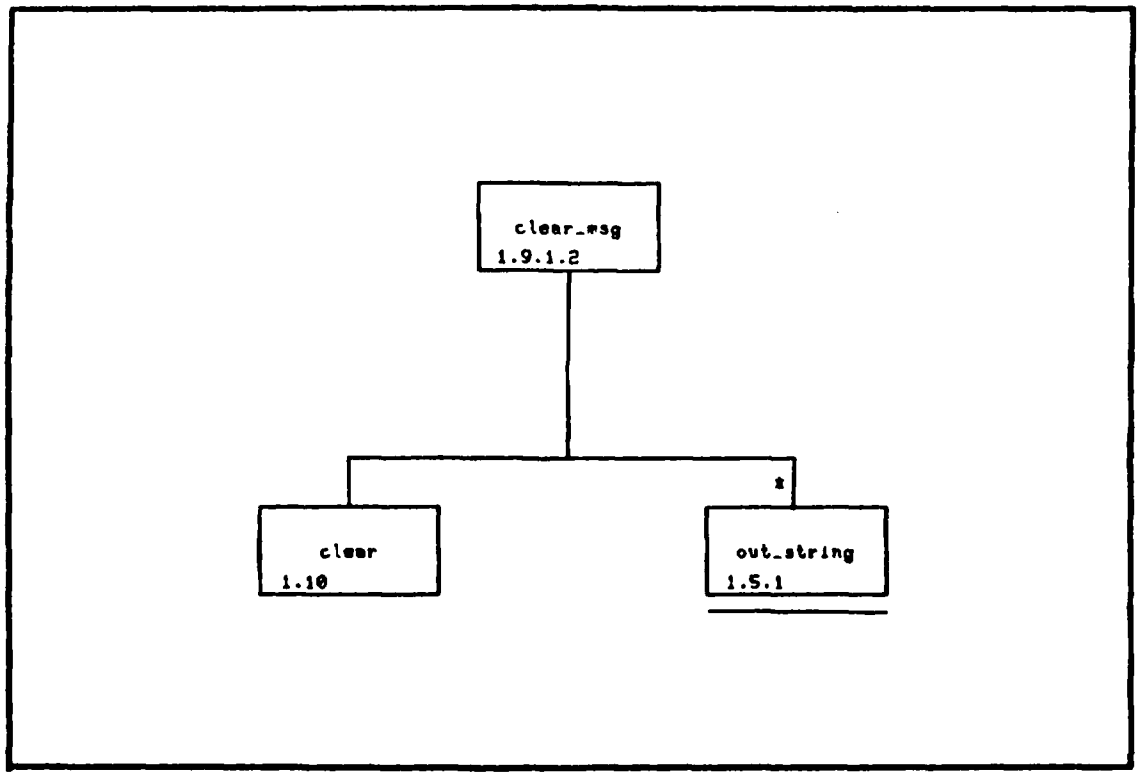

prompt_cmd
1.7
highlight
1.7.1
gotoxy
1.1
out_string
1.5.1
nohighlight
1.7.2

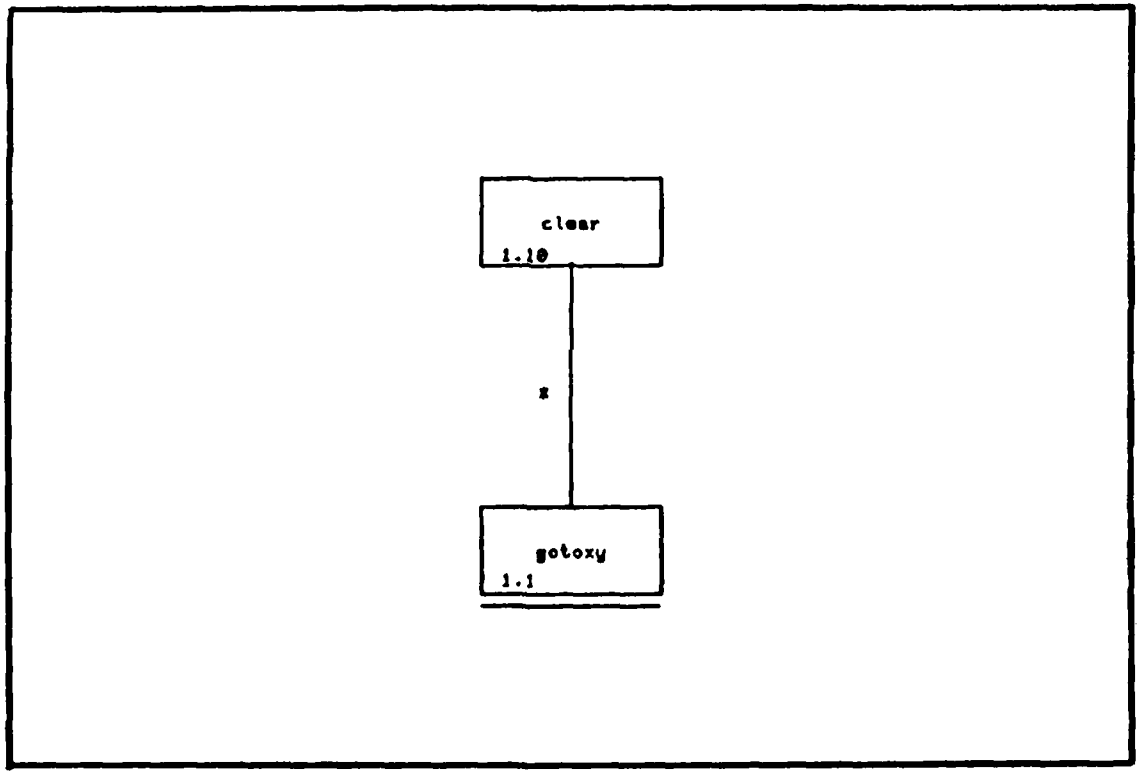

instruction
1.8
*
out_string
1.5.1

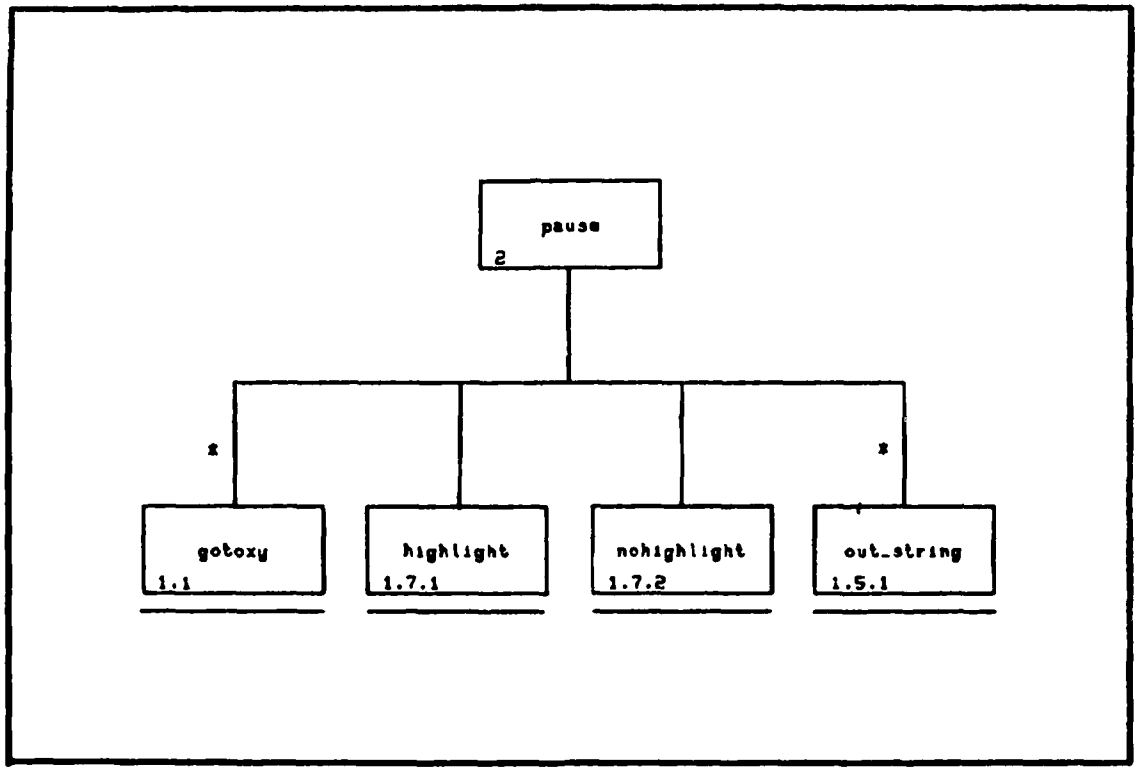

proces_error
1.9
x
x
gotoxy
1.1
pause
2
displa_
commandword
1.6
1
highlight
1.7.1
nohighlight
1.7.2
disp_msg
1.9.1
1. msg_num

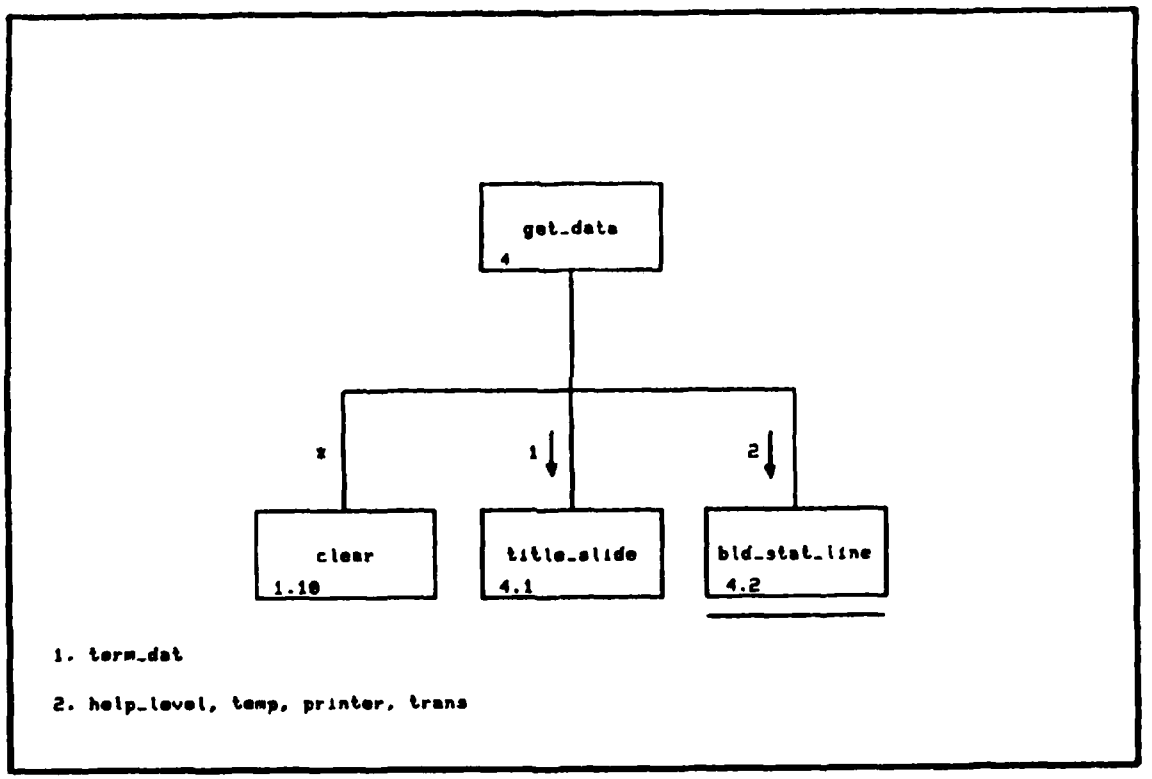

disp_msg
1.9.1
*
1
*
2
gotoxy
1.1
disp_line
1.9.1.1
disp_msg
1.9.1
clear
1.10
clear_msg
1.9.1.2
1. rec_num
2. msg_num


disp_line
1.9.1.1
*
out_string
1.5.1


clear_msg
1.9.1.2
clear
1.10
out_string
1.5.1

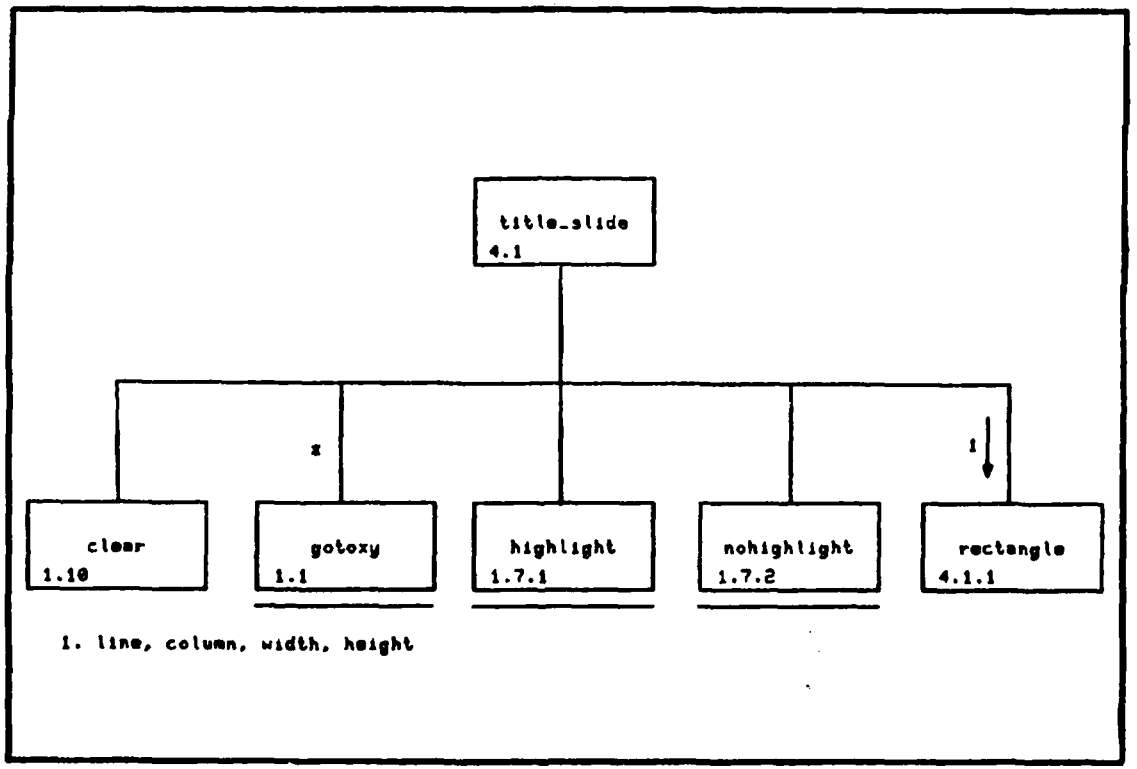

clear
1.10
x
gotoxy
1.1

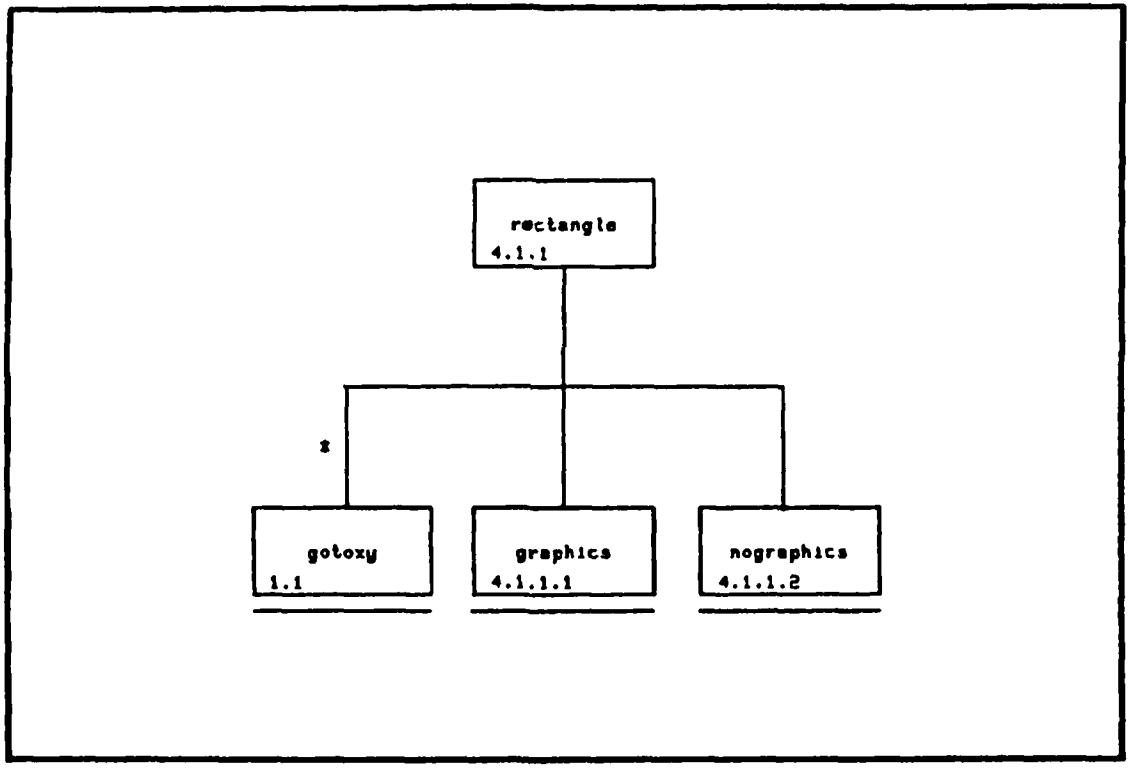

pause
2
x
x
gotoxy
1.1
highlight
1.7.1
nohighlight
1.7.2
out_string
1.5.1

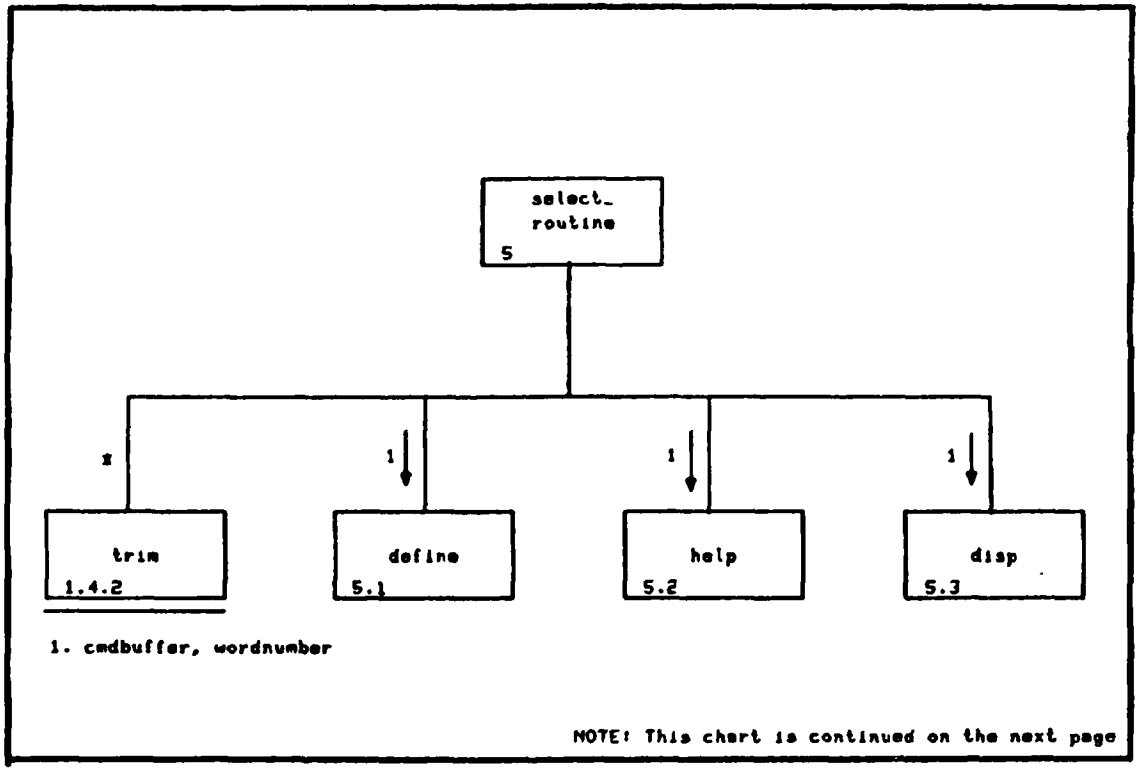


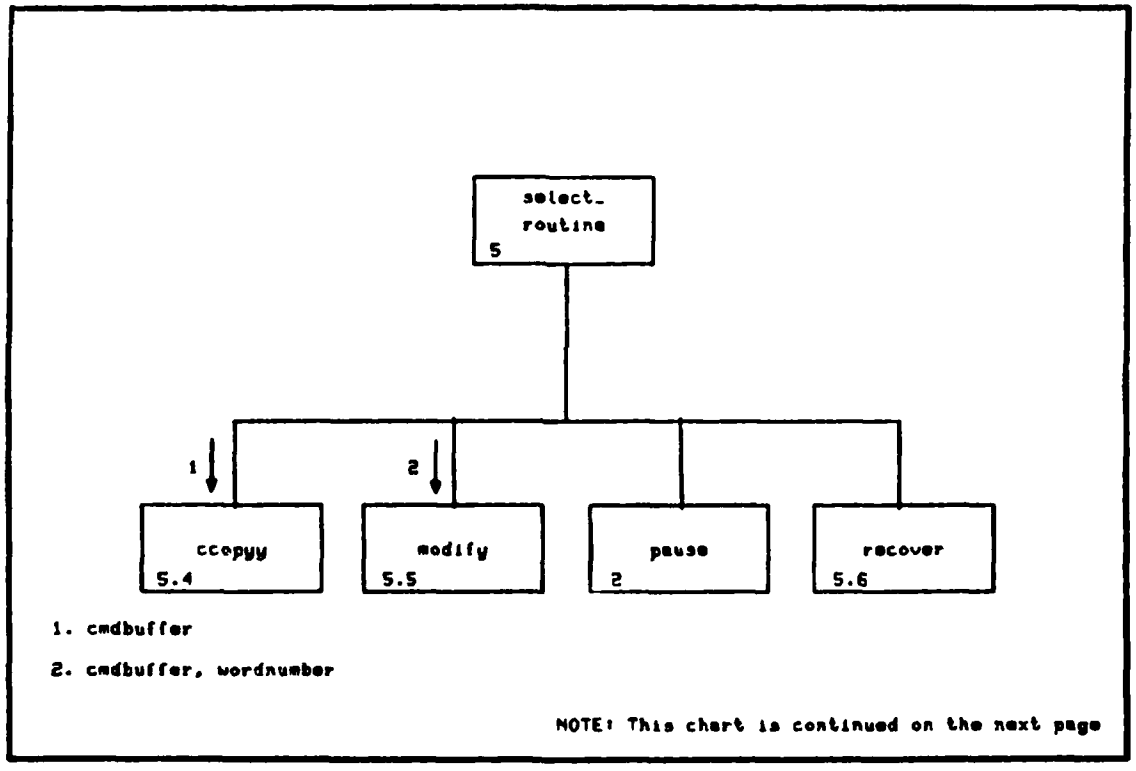
1. term_dat

2. help_level, temp, printer, trans


title_slide
4.1
x
1
clear
1.10
gotoxy
1.1
highlight
1.7.1
nohighlight
1.7.2
rectangle
4.1.1
1. line, column, width, height


rectangle
4.1.1
x
gotoxy
1.1
graphics
4.1.1.1
nographics
4.1.1.2

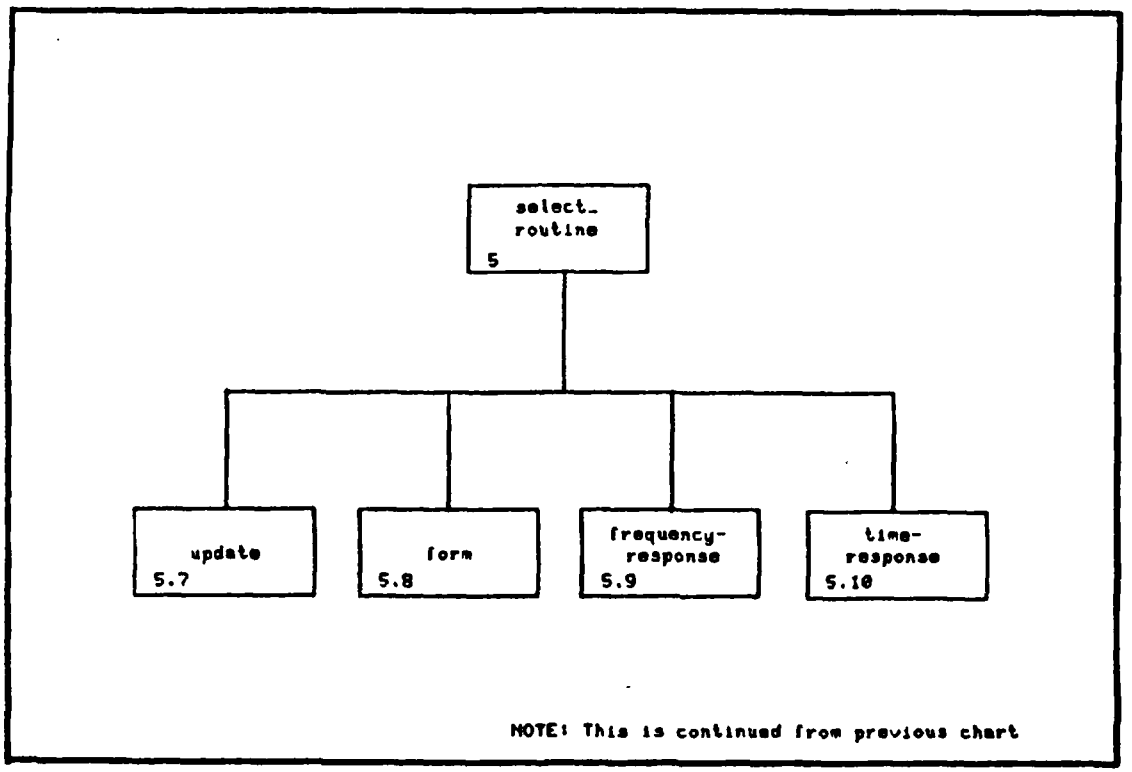

select_
routine
5
x
1
1
1
trim
1.4.2
define
5.1
help
5.2
disp
5.3
1. cmdbuffer, wordnumber
NOTE: This chart is continued on the next page

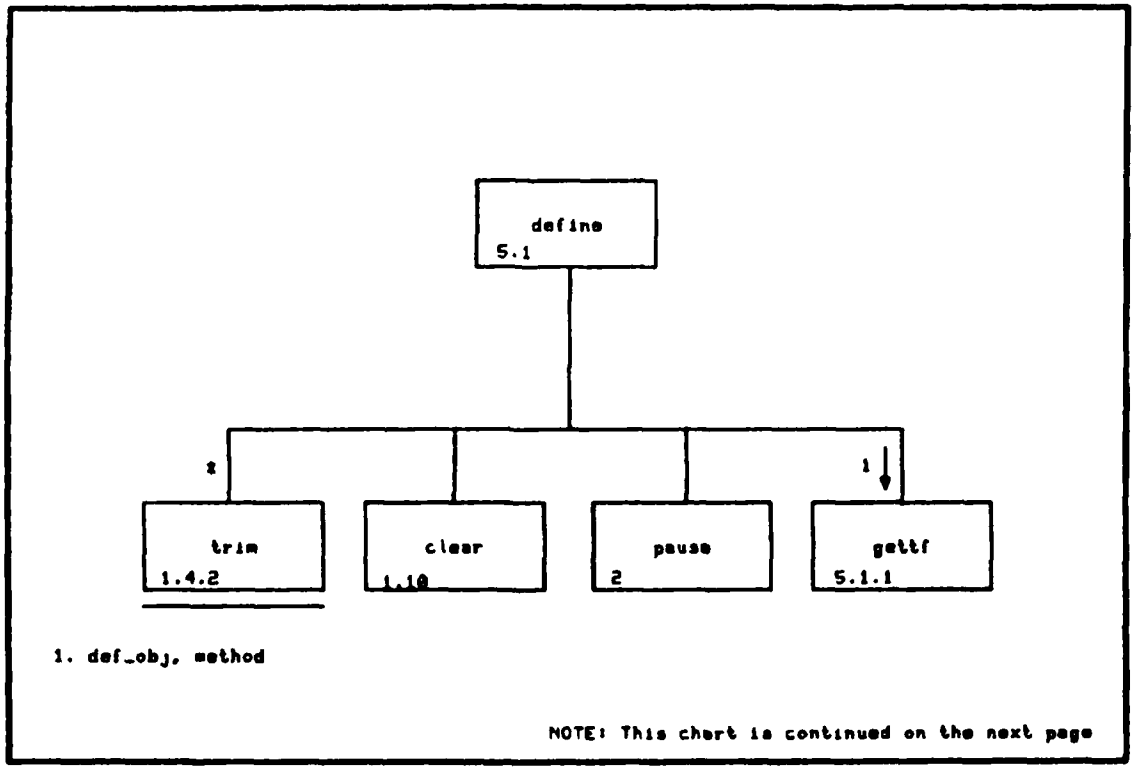


1. cmdbuffer

2. cmdbuffer, wordnumber

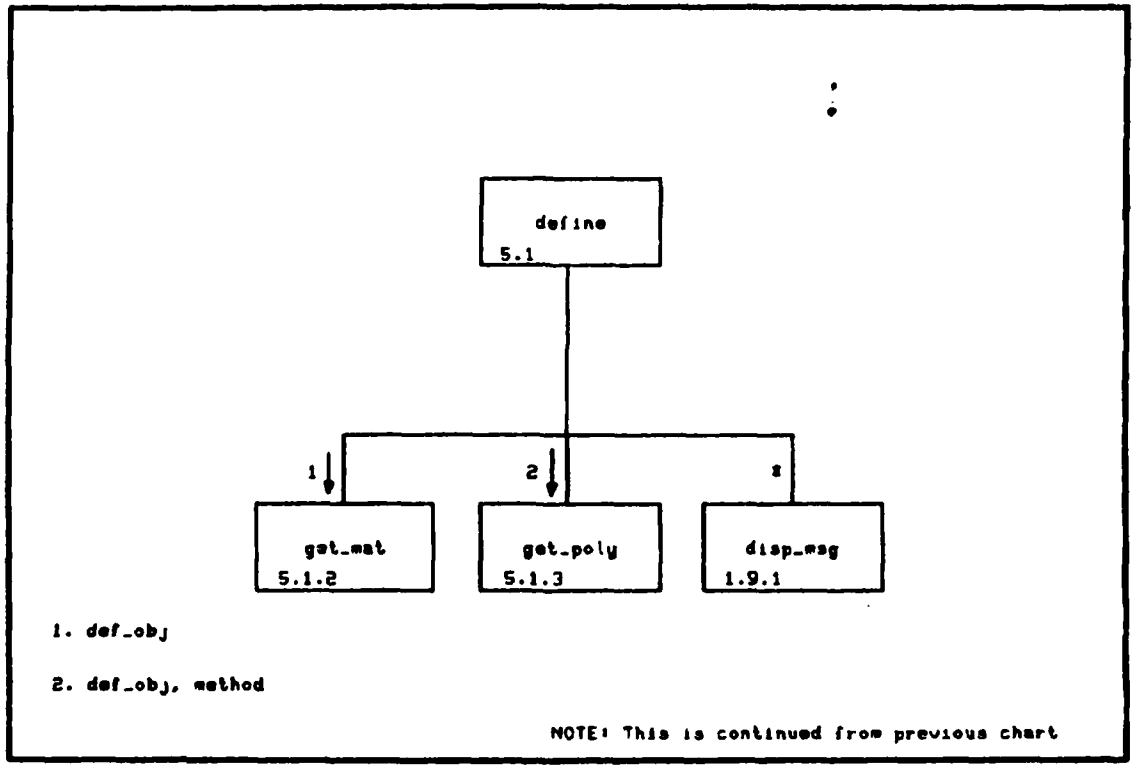
NOTE: This chart is continued on the next page


select_
routine
5
update
5.7
form
5.8
frequency-
response
5.9
time-
response
5.10
NOTE: This is continued from previous chart



1. def_obj, method

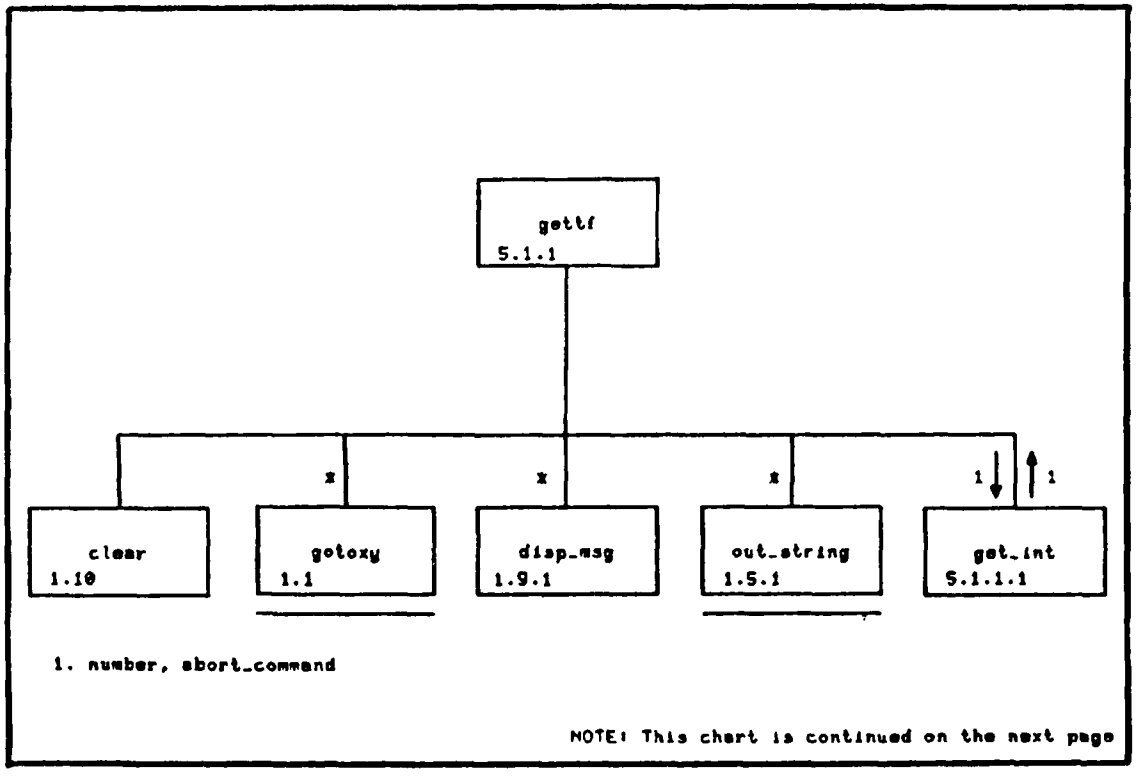
NOTE: This chart is continued on the next page

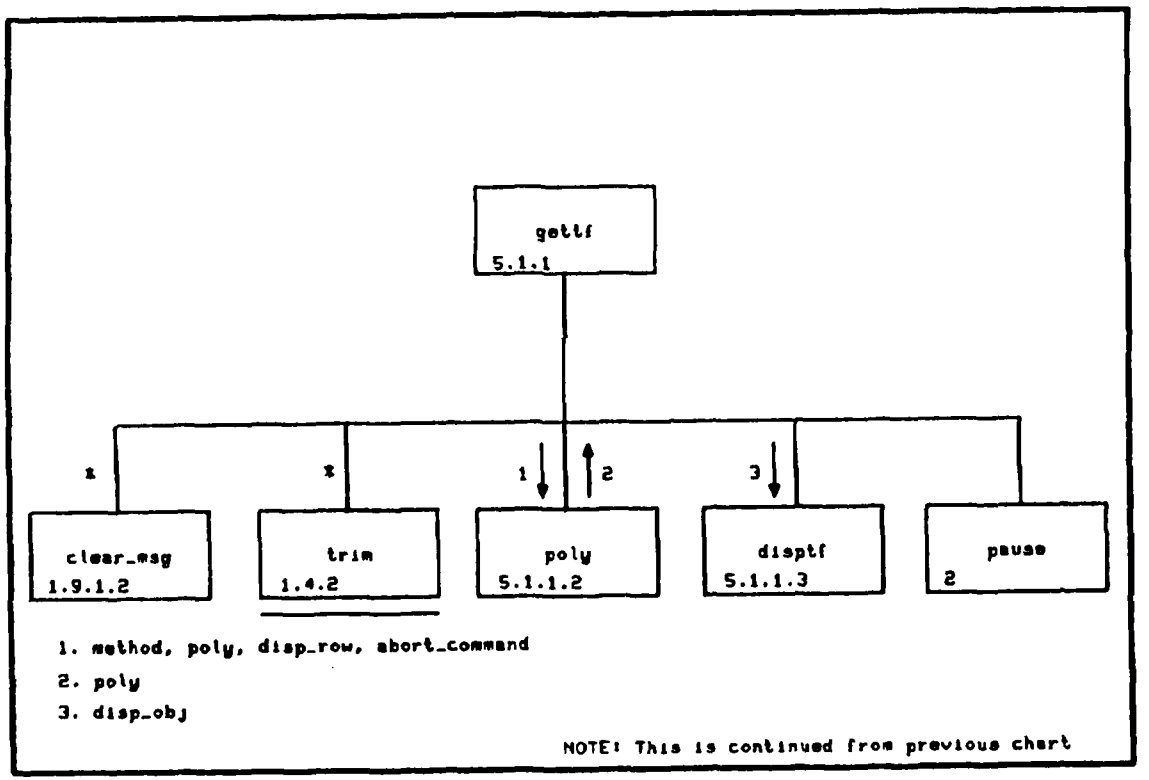


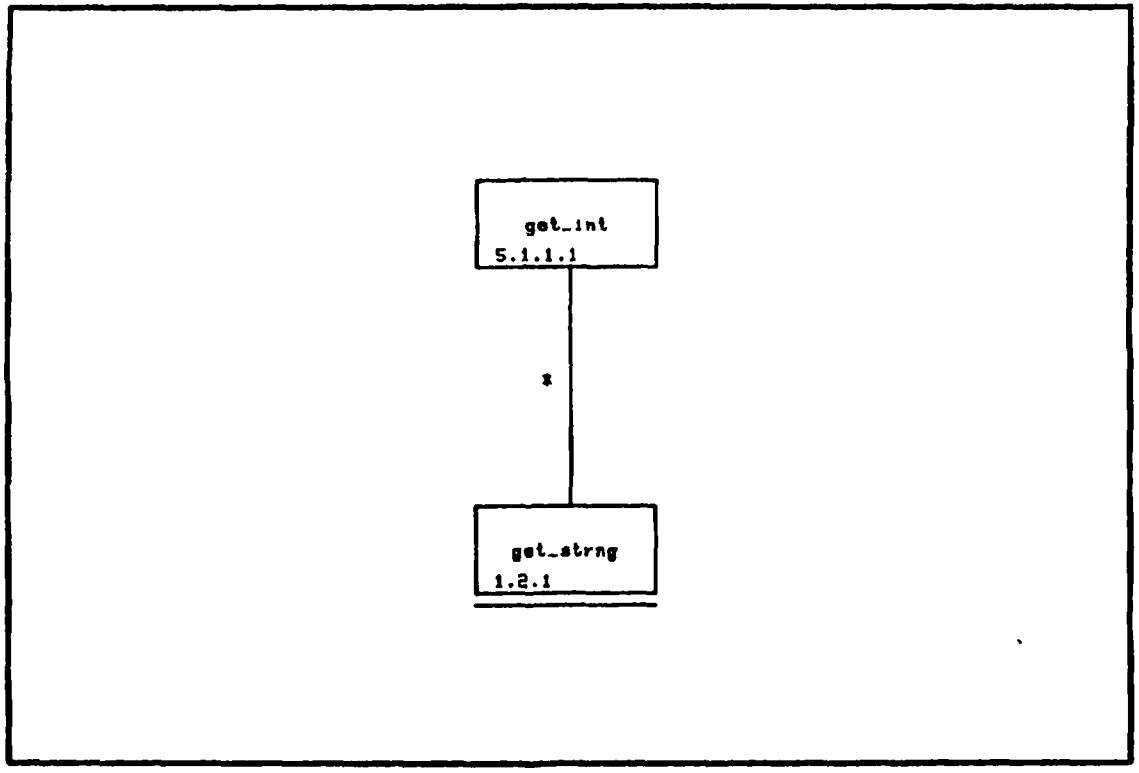
1. def_obj

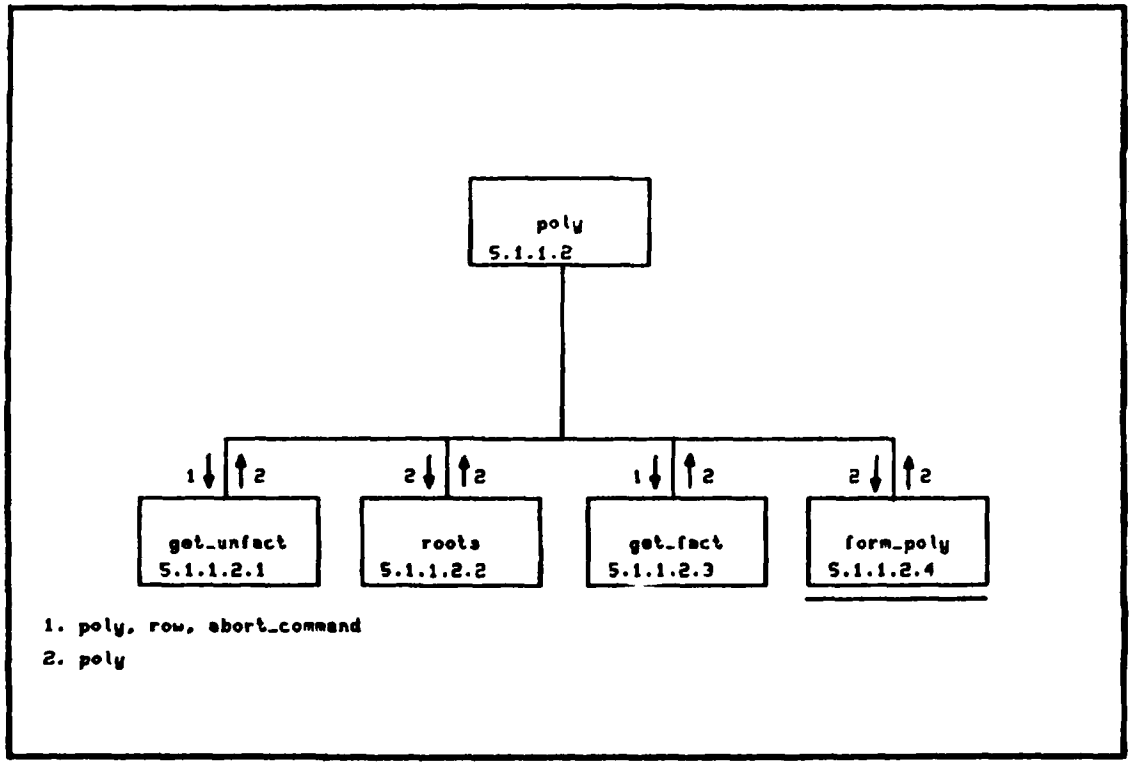
2. def_obj, method

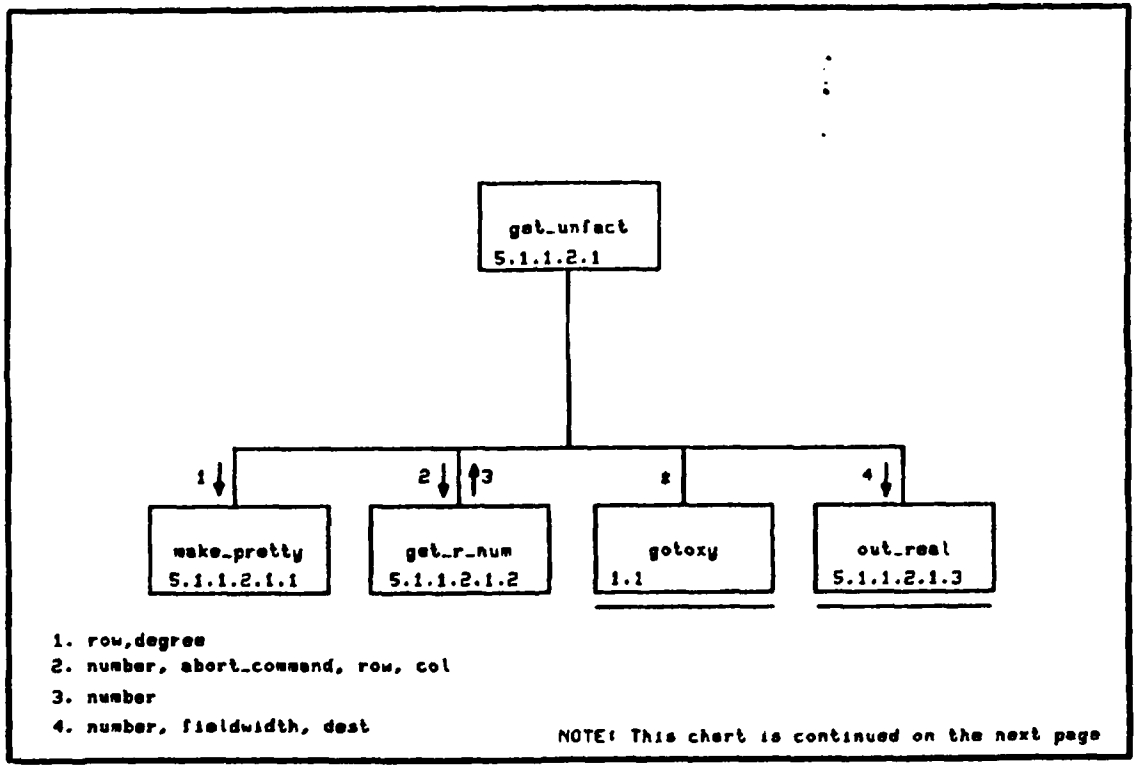
NOTE: This is continued from previous chart

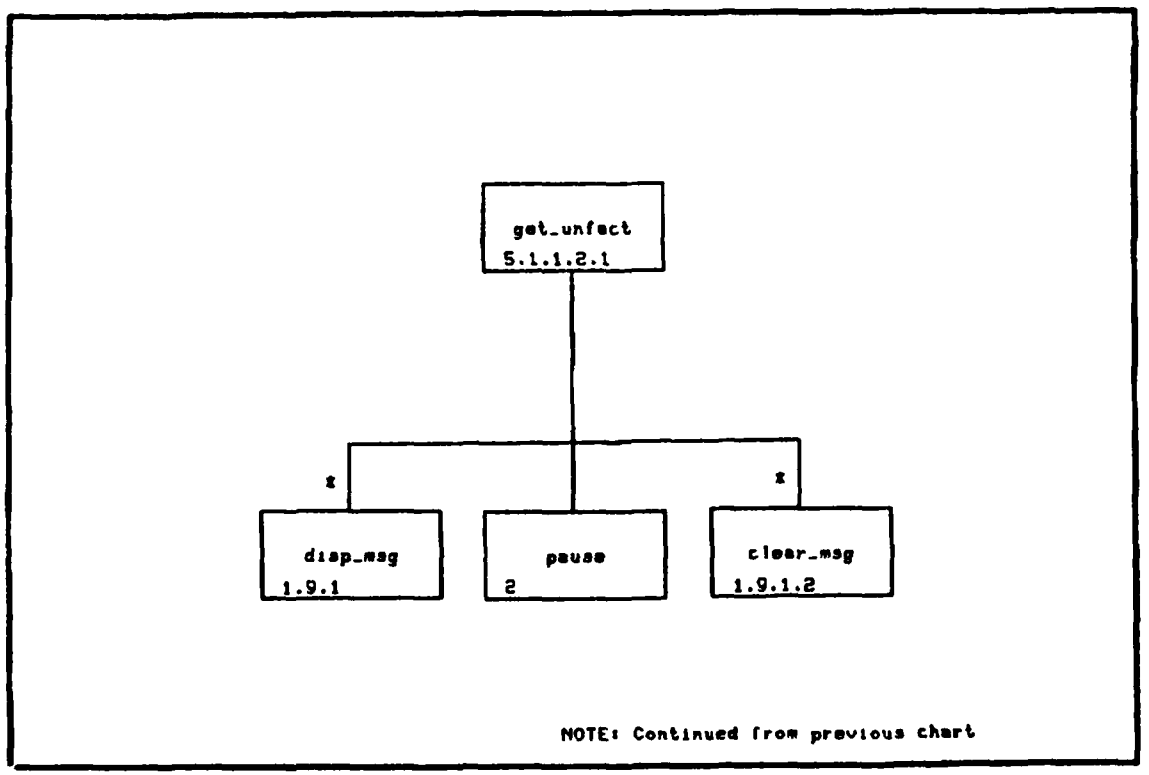

gettf
5.1.1
x
x
x
1
1
clear
1.10
getxy
1.1
disp_msg
1.9.1
out_string
1.5.1
get_int
5.1.1.1
1. number, abort_command
NOTE: This chart is continued on the next page

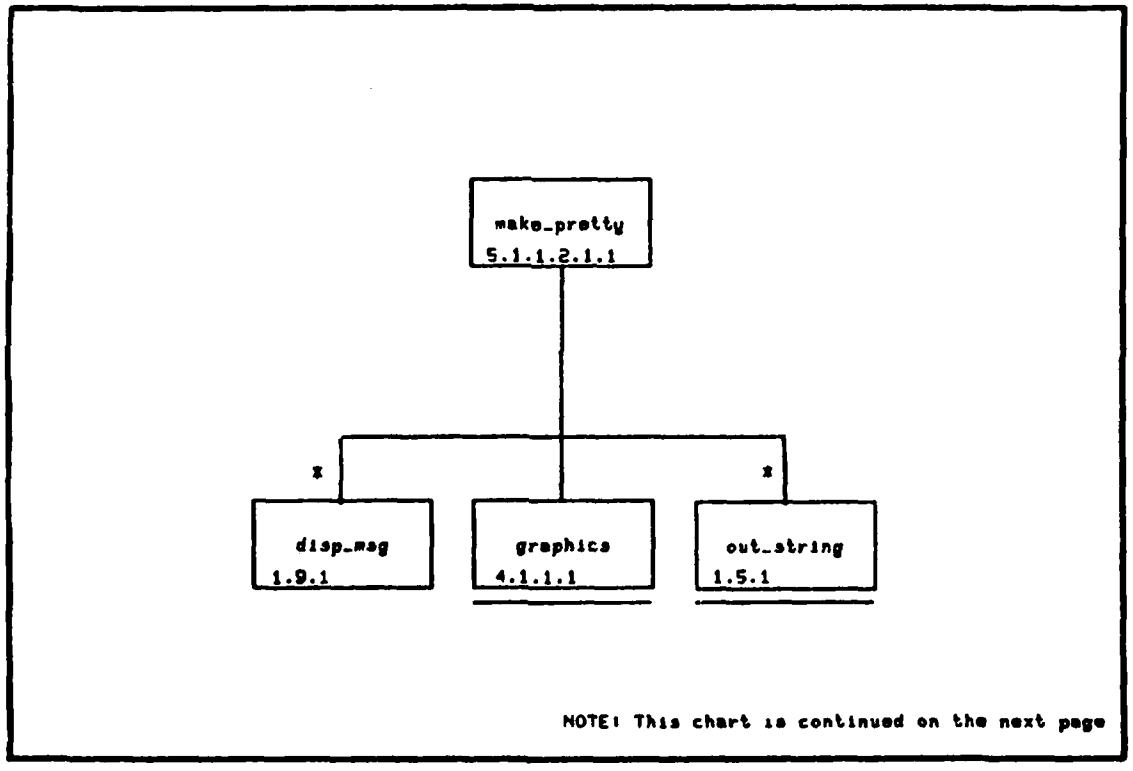

gettf
5.1.1
1
1
1
2
3
clear_msg
1.9.1.2
trim
1.4.2
poly
5.1.1.2
disptf
5.1.1.3
pause
2
1. method, poly, disp_row, abort_command
2. poly
3. disp_obj
NOTE: This is continued from previous chart

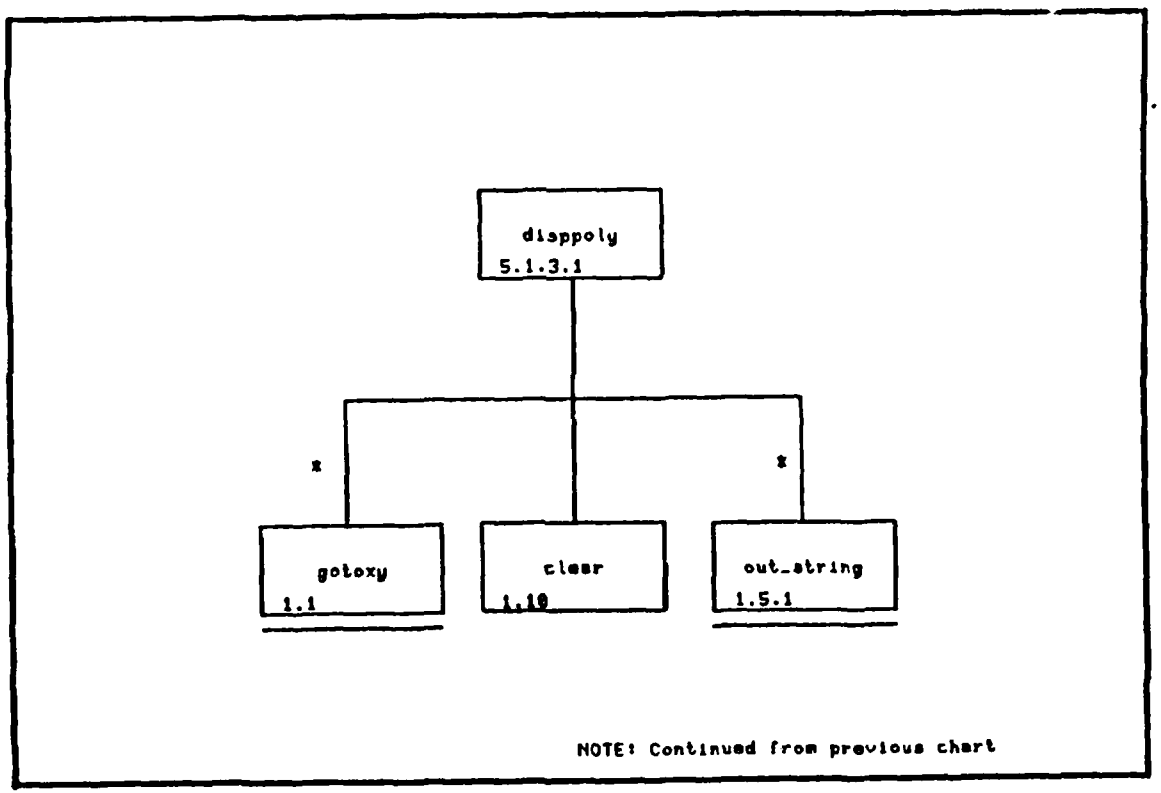

get_int
5.1.1.1
1
get_strng
1.2.1

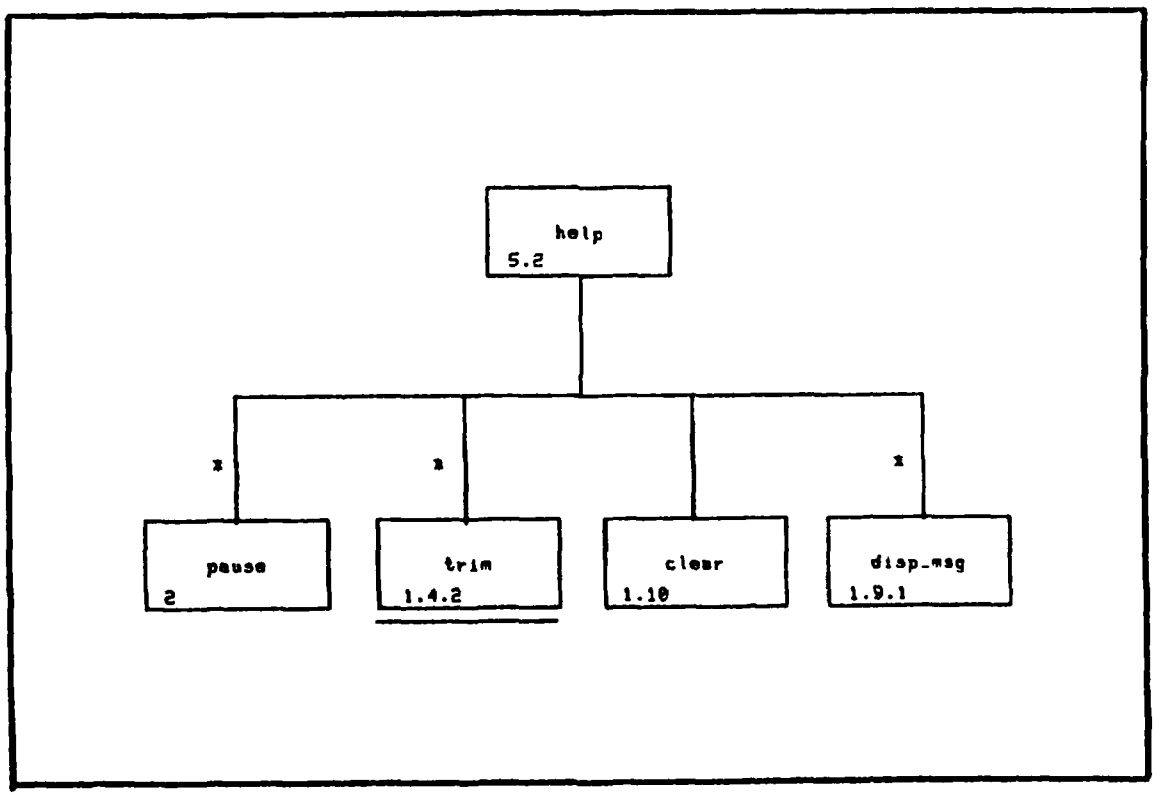

poly
5.1.1.2
1
2
2
2
1
2
2
2
get_unfact
5.1.1.2.1
roots
5.1.1.2.2
get_fact
5.1.1.2.3
form_poly
5.1.1.2.4
1. poly, row, abort_command
2. poly

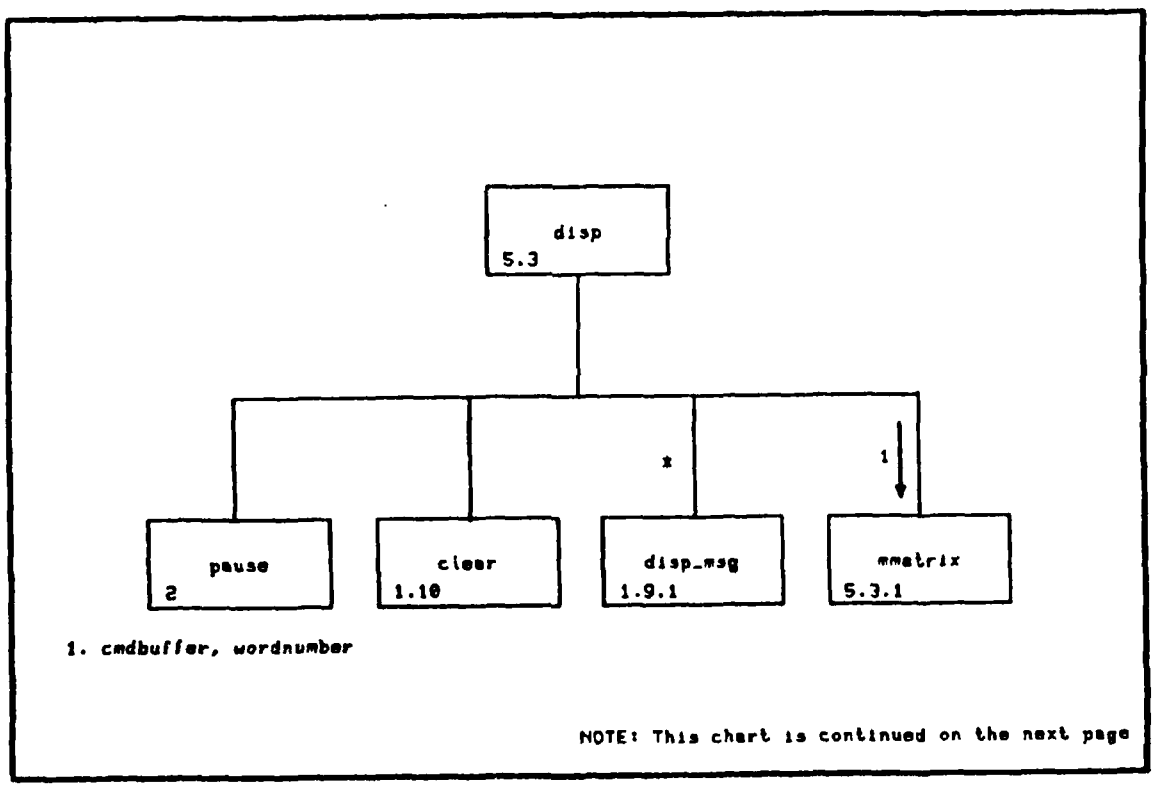

get_unfact
5.1.1.2.1
1
2
3
4
make_pretty
5.1.1.2.1.1
get_r_num
5.1.1.2.1.2
gotoxy
1.1
out_real
5.1.1.2.1.3
1. row,degree
2. number, abort_command, row, col
3. number
4. number, fieldwidth, dest
NOTE: This chart is continued on the next page


get_unfact
5.1.1.2.1
x
x
disp_msg
1.9.1
pause
2
clear_msg
1.9.1.2
NOTE: Continued from previous chart


make_pretty
5.1.1.2.1.1
x
x
disp_msg
1.9.1
graphics
4.1.1.1
out_string
1.5.1
NOTE: This chart is continued on the next page

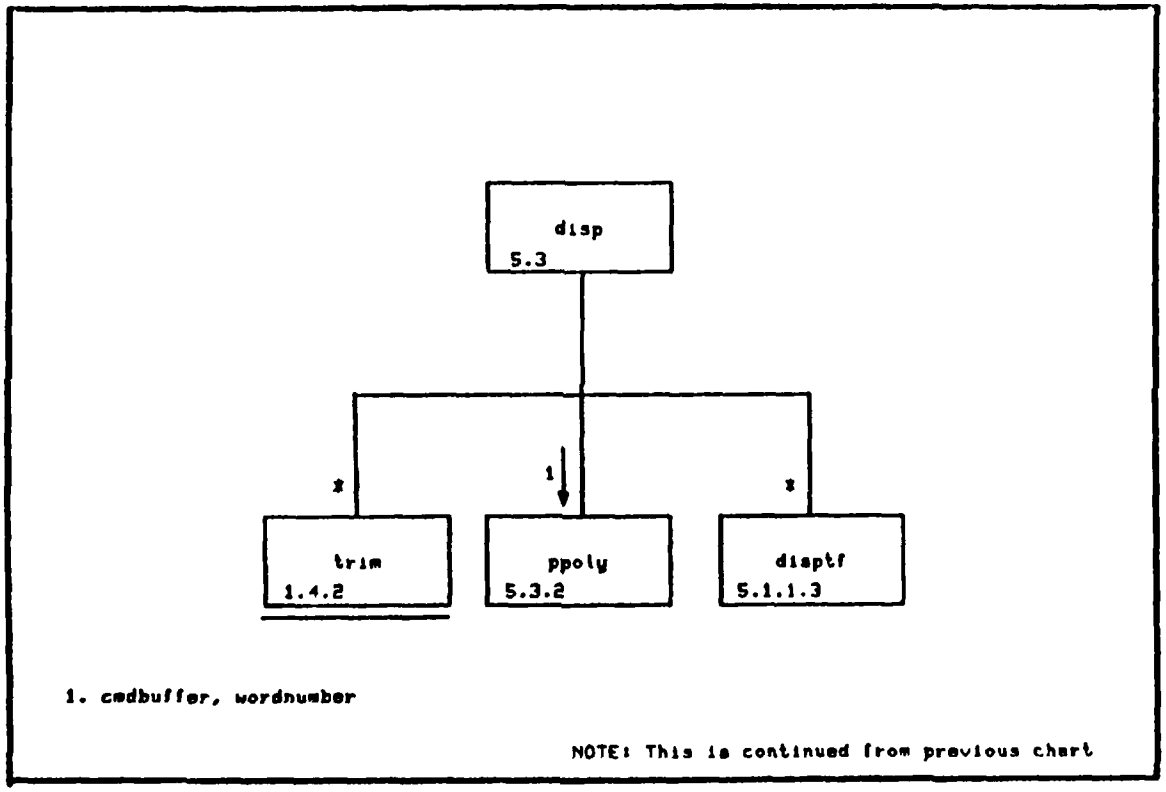

disppoly
5.1.3.1
x
x
gotoxy
1.1
clear
1.10
out_string
1.5.1
NOTE: Continued from previous chart


help
5.2
x
x
x
pause
2
trim
1.4.2
clear
1.10
disp_msg
1.9.1



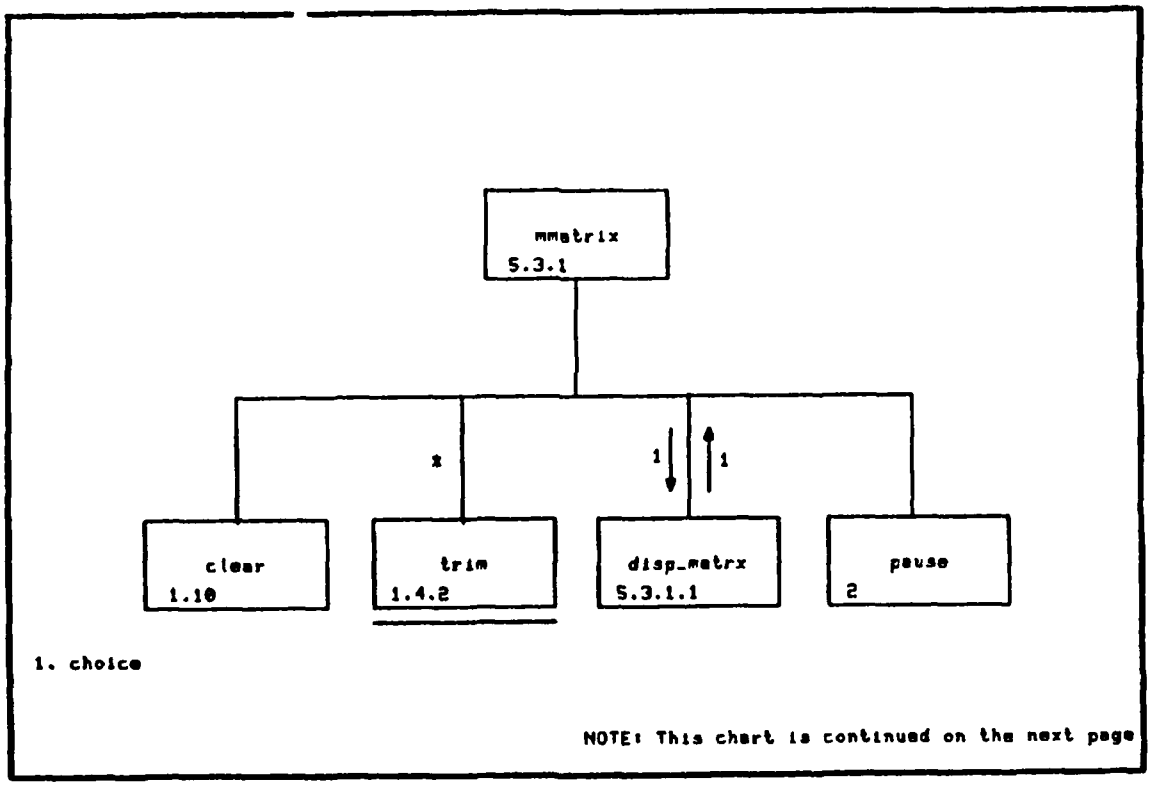
1. *cmdbuffer, wordnumber*

NOTE: This chart is continued on the next page



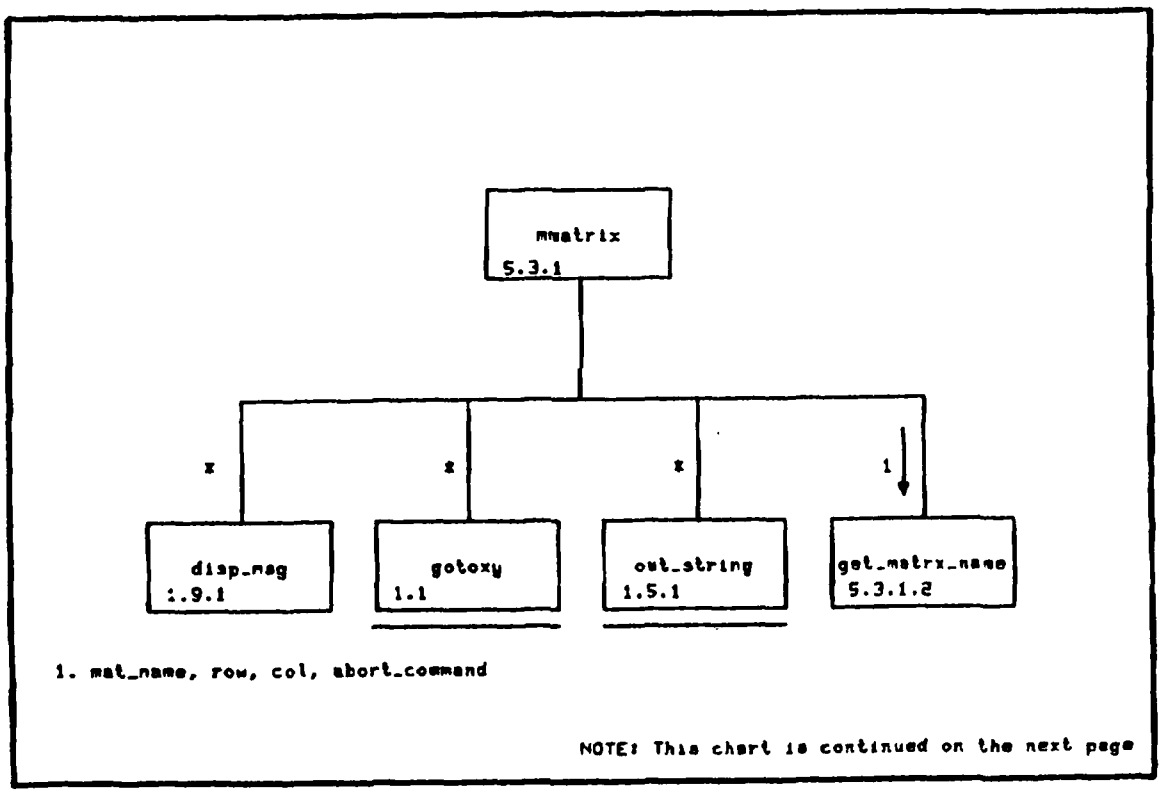
1. cmdbuffer, wordnumber

NOTE: This is continued from previous chart



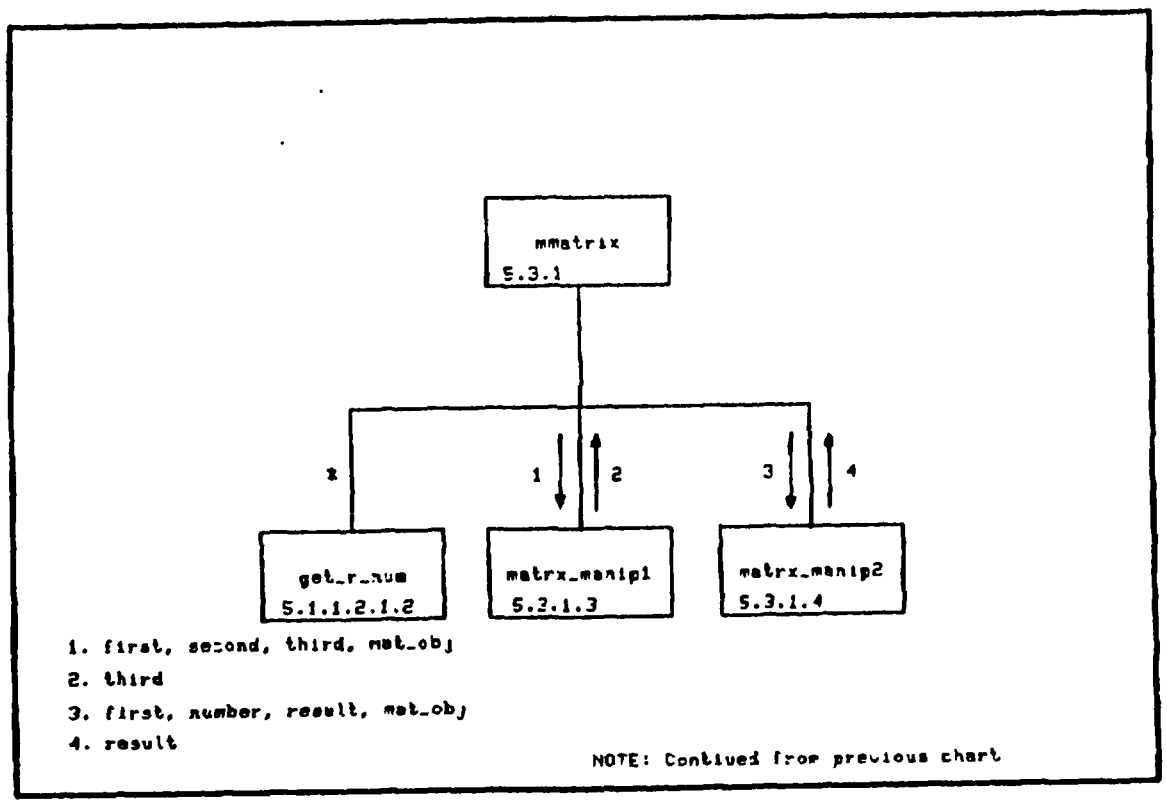
1. choice

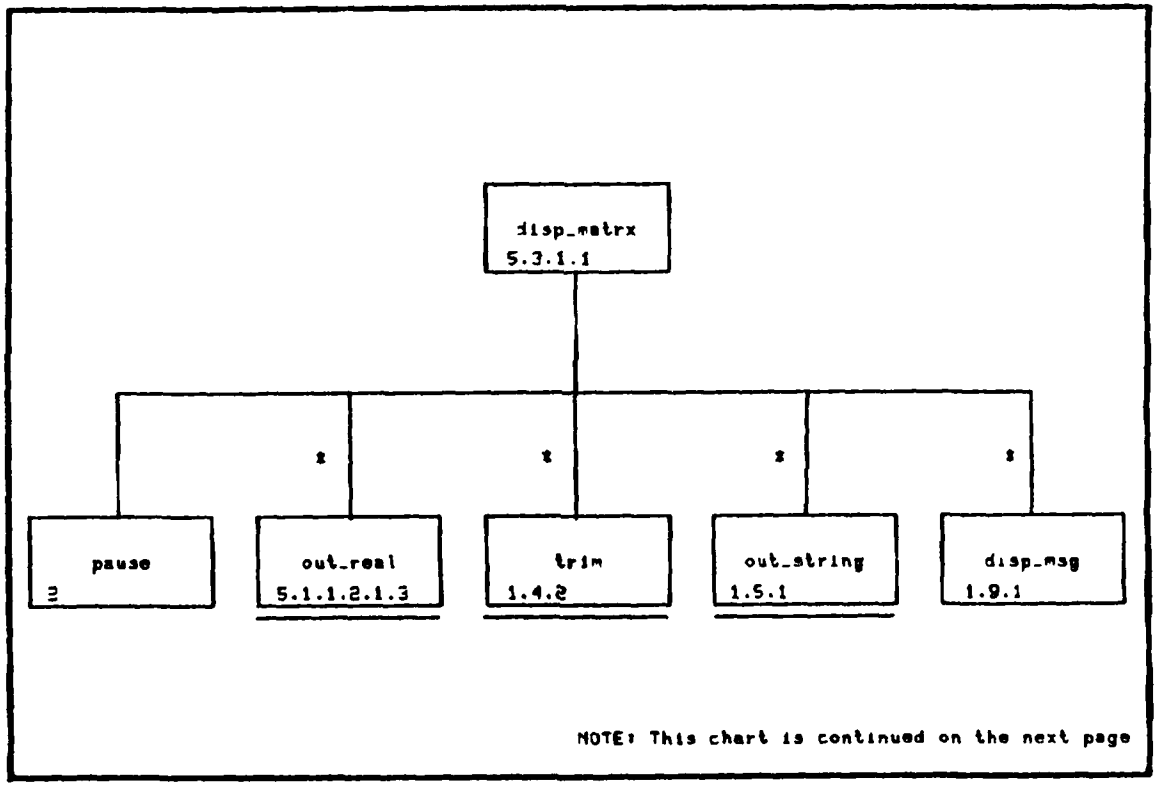
NOTE: This chart is continued on the next page

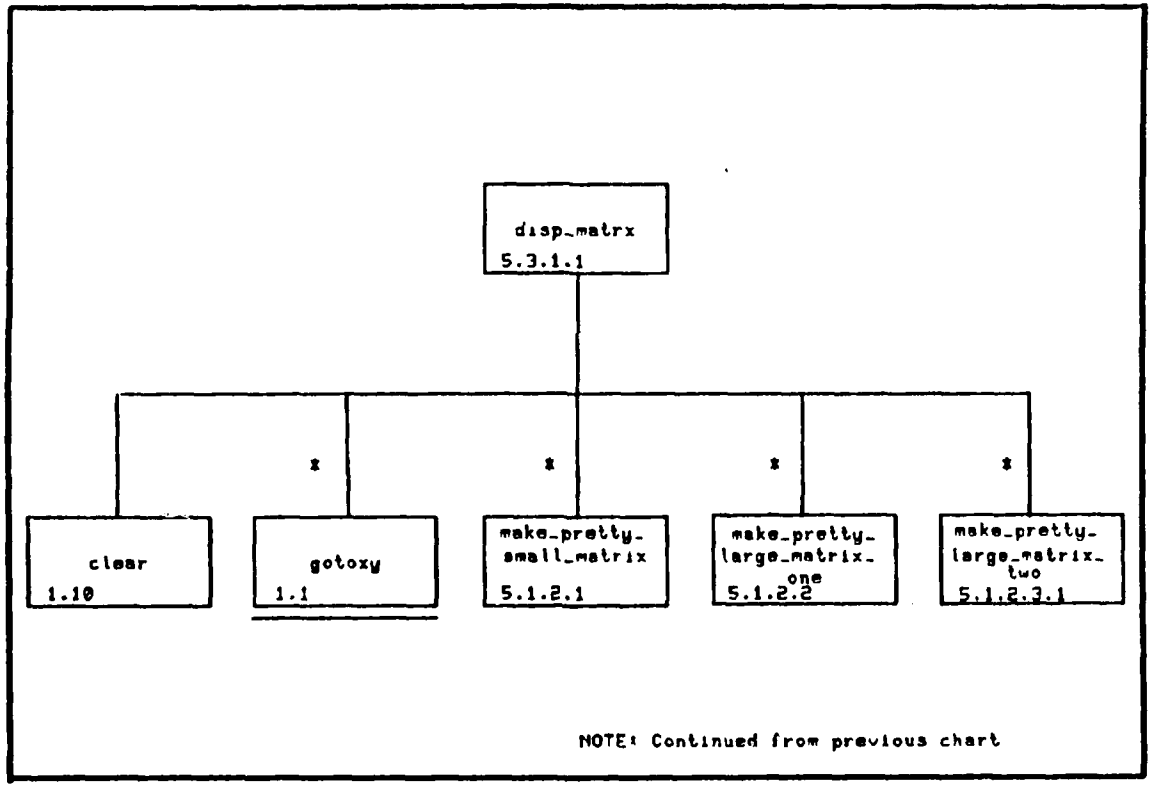


1. mat_name, row, col, abort_command

NOTE: This chart is continued on the next page

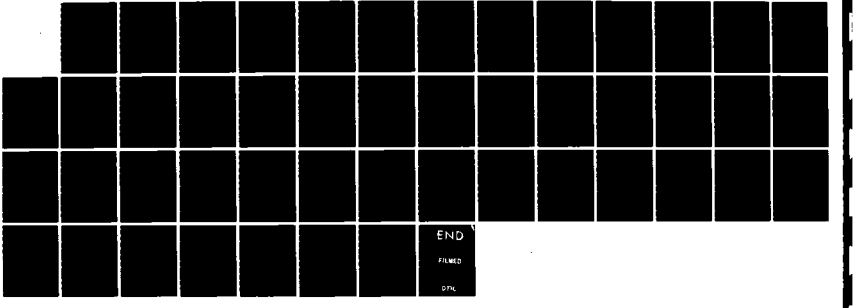

mmatrix
5.3.1
1
1
2
3
4
get_r_num
5.1.1.2.1.2
matrx_manip1
5.2.1.3
matrx_manip2
5.3.1.4
1. first, second, third, mat_obj
2. third
3. first, number, result, mat_obj
4. result
NOTE: Contiued from previous chart

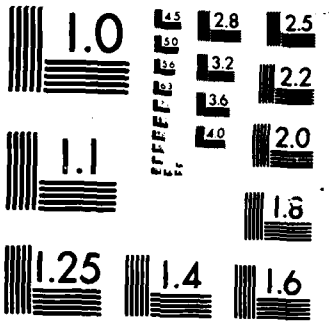

disp_matrx
5.3.1.1
pause
3
out_real
5.1.1.2.1.3
trim
1.4.2
out_string
1.5.1
disp_msg
1.9.1
NOTE: This chart is continued on the next page

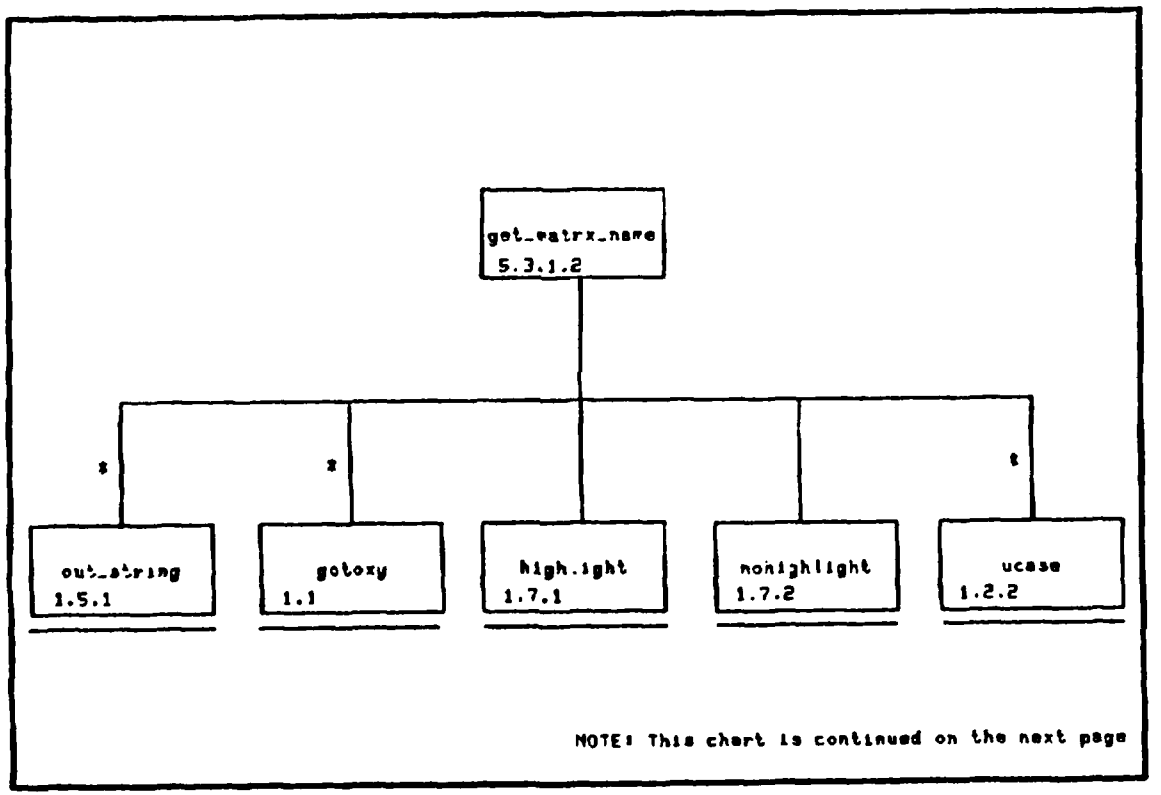

disp_matrx
5.3.1.1
clear
1.10
gotoxy
1.1
make_pretty_
small_matrix
5.1.2.1
make_pretty_
large_matrix_
one
5.1.2.2
make_pretty_
large_matrix_
two
5.1.2.3.1
NOTE: Continued from previous chart

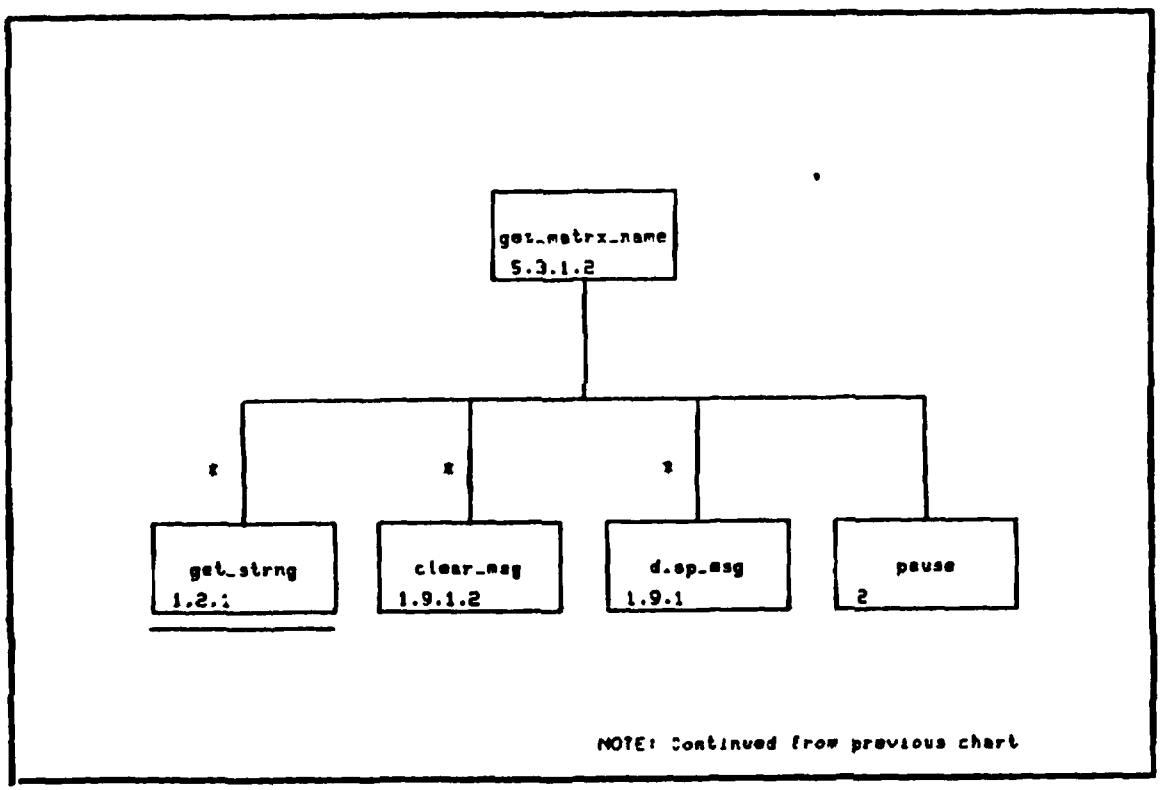
AD-A163 940 DEVELOPMENT OF A COMPUTER AIDED DESIGN PACKAGE FOR CONTROL SYSTEM DESIGN A.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 3/3

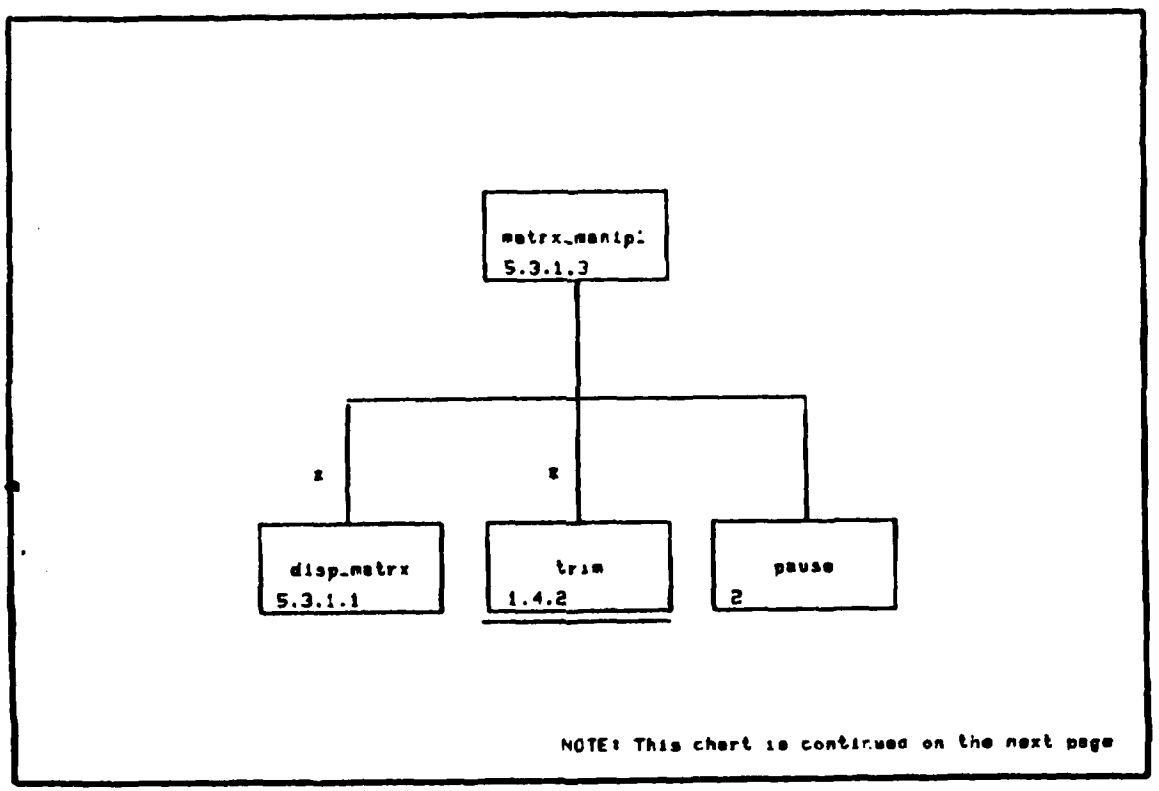
UNCLASSIFIED S K MASHIKO ET AL. DEC 85 F/G 9/2 NL

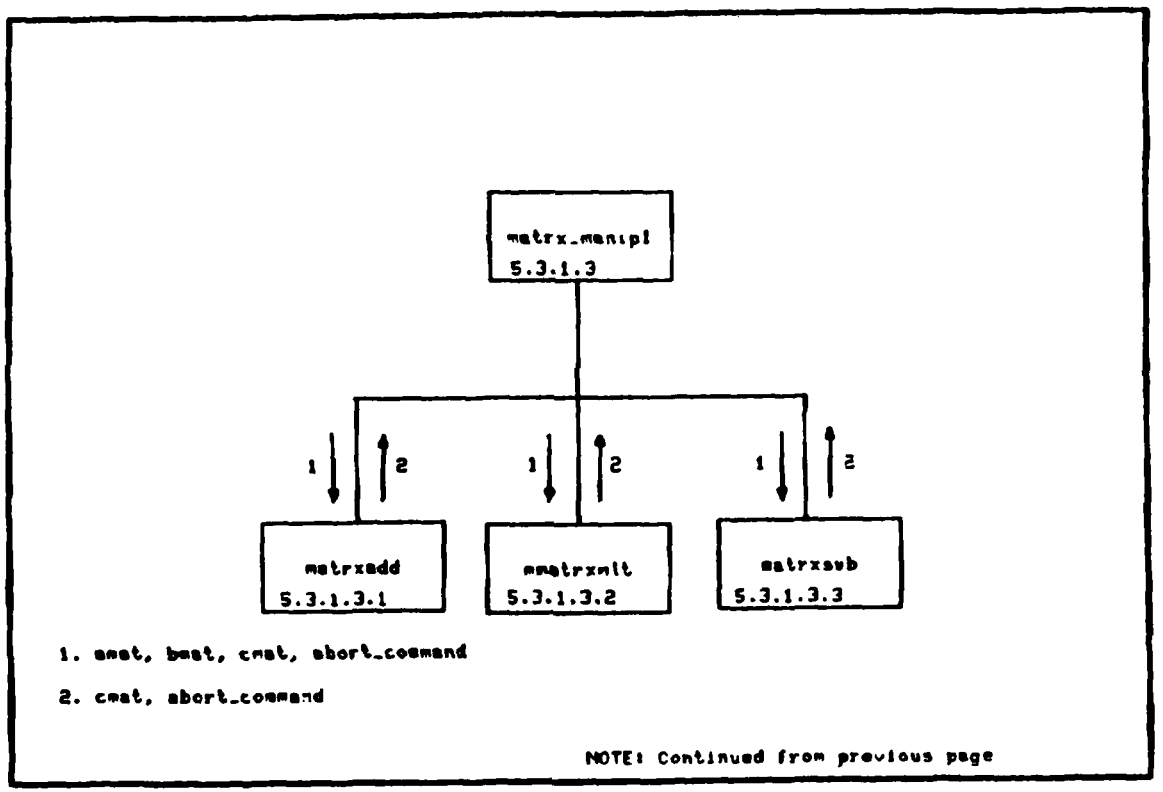





get_vatrx_name
5.3.1.2
s
s
s
out_string
1.5.1
gotoxy
1.1
highlight
1.7.1
nohighlight
1.7.2
ucase
1.2.2
NOTE: This chart is continued on the next page


get_matrx_name
5.3.1.2
x
x
x
get_strng
1.2.1
clear_msg
1.9.1.2
d.sp_msg
1.9.1
pause
2
NOTE: Continued from previous chart

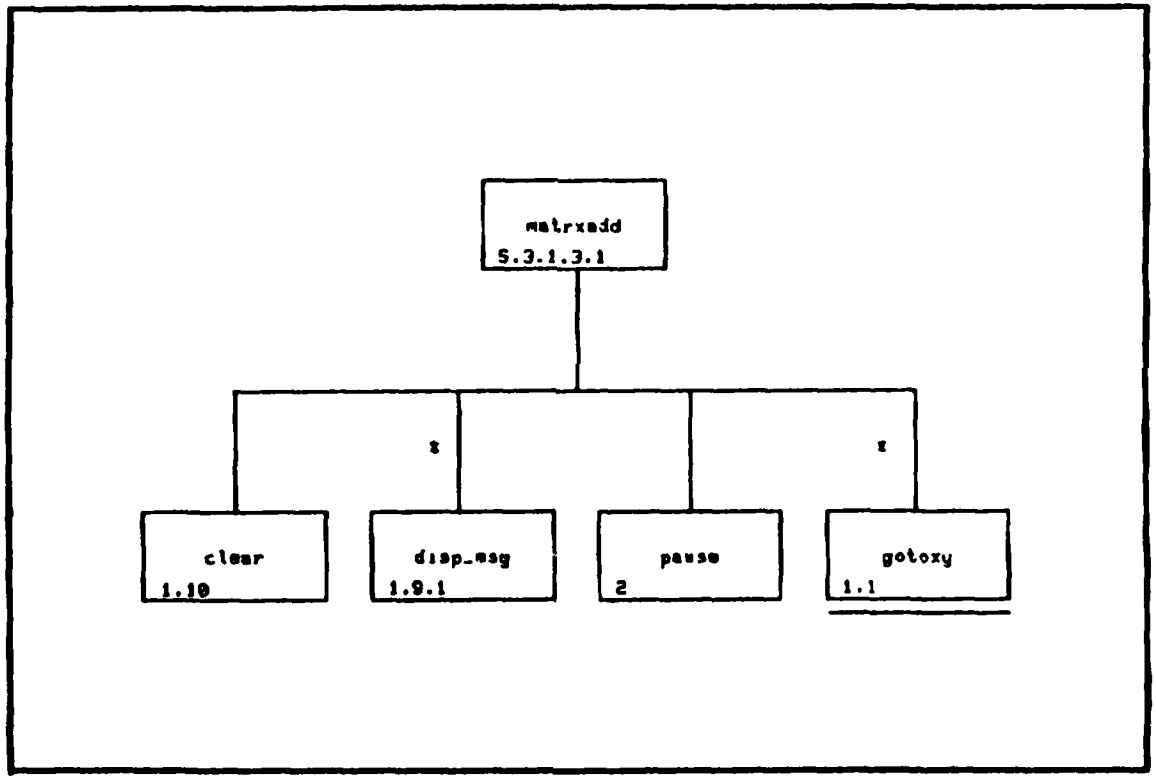

matrx_manip
5.3.1.3
2
2
disp_matrx
5.3.1.1
trim
1.4.2
pause
2
NOTE: This chart is continued on the next page

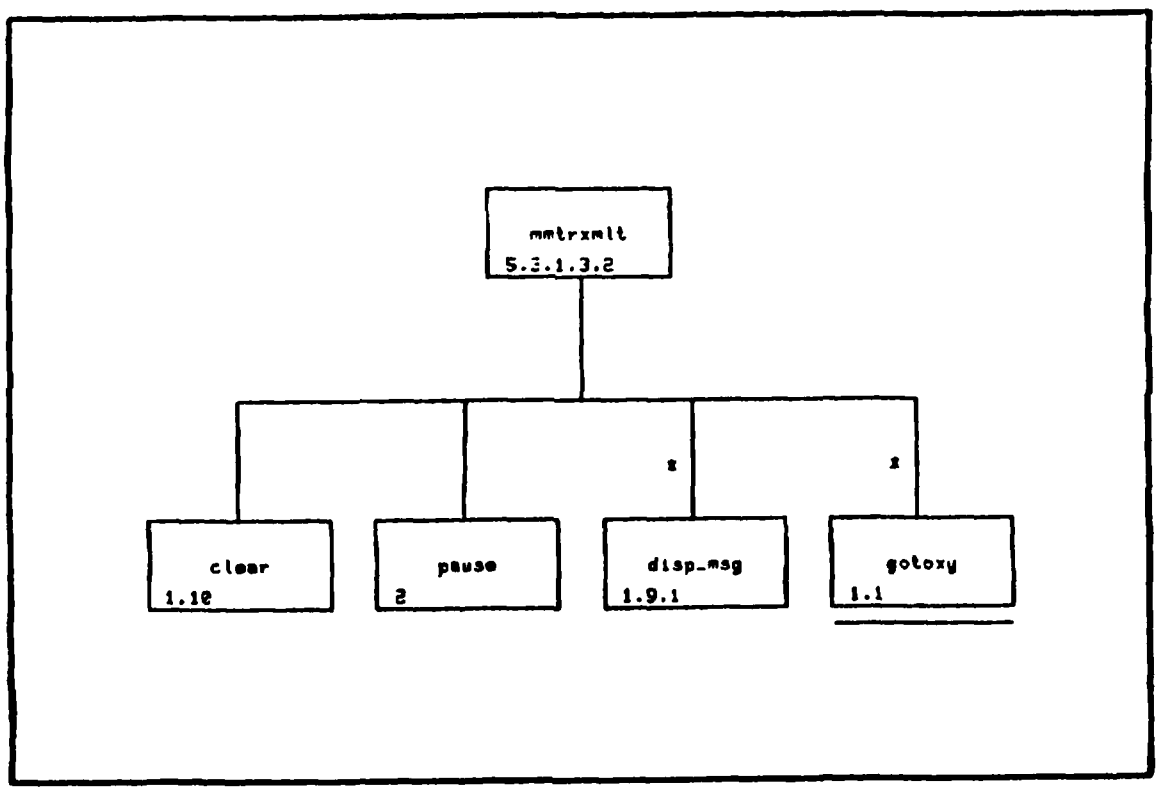


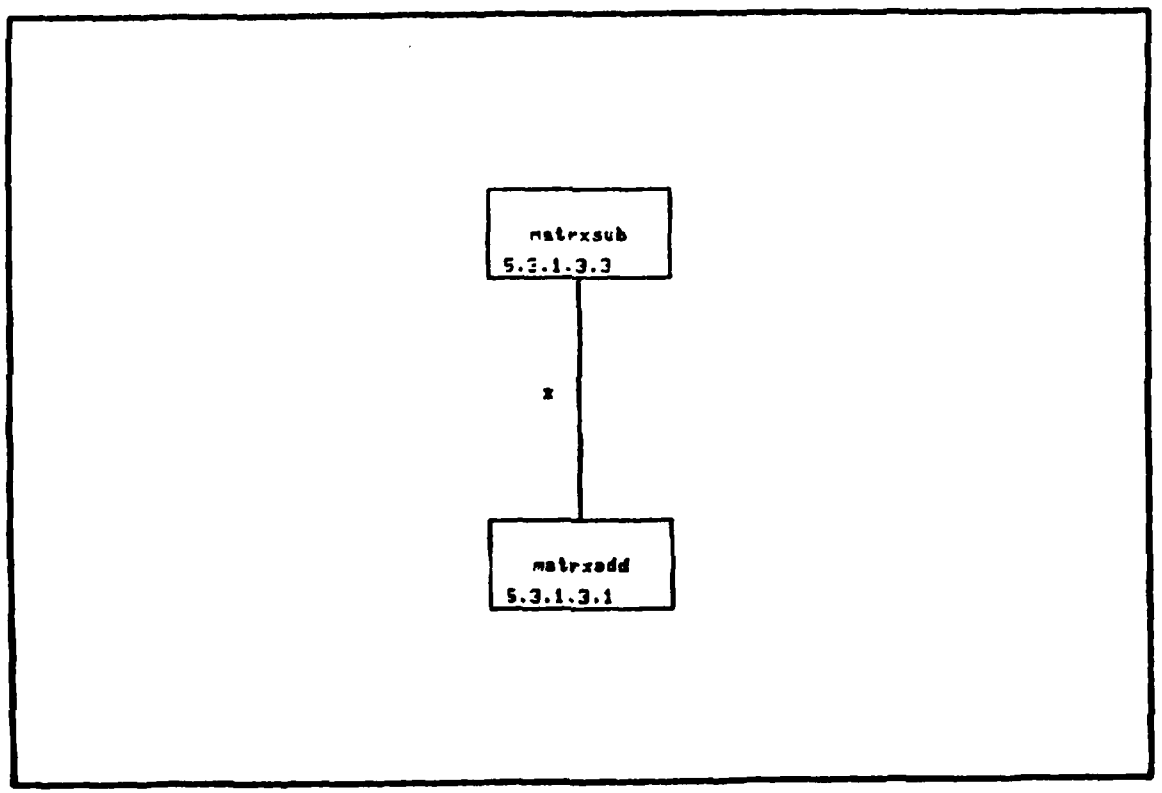
1. amat, bmat, cmat, abort_command

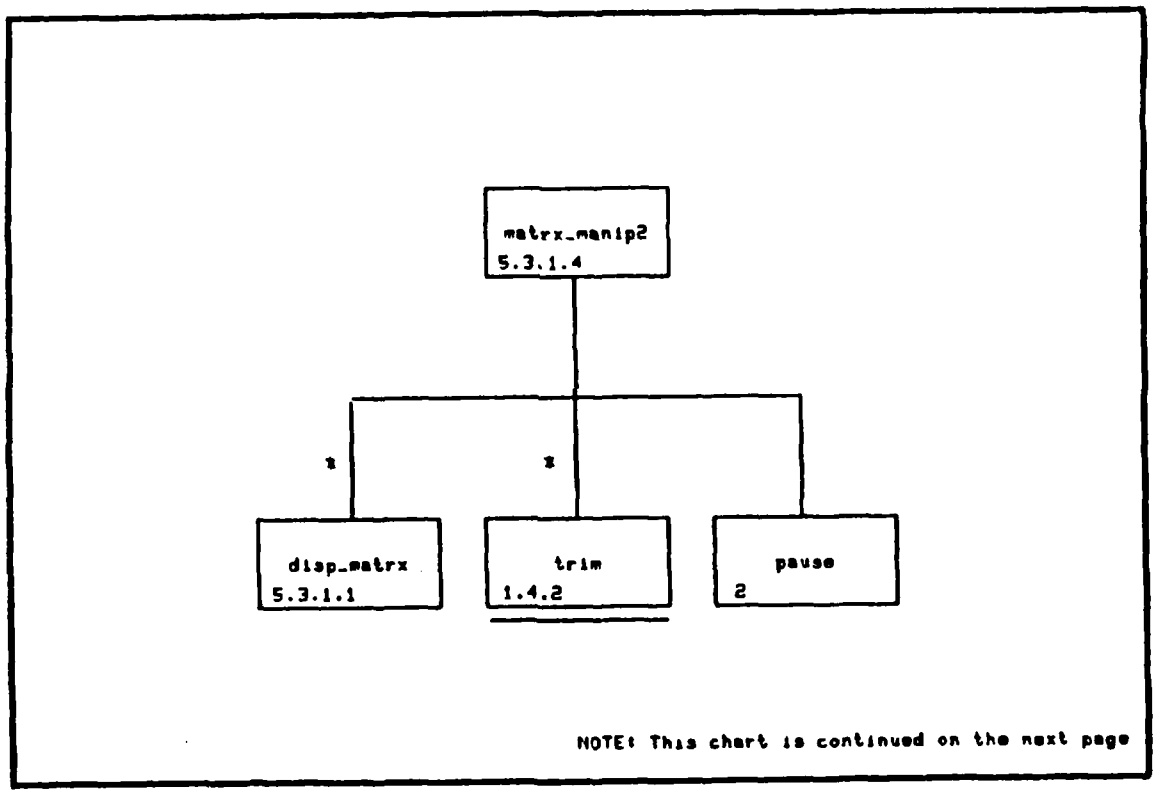
2. cmat, abort_command

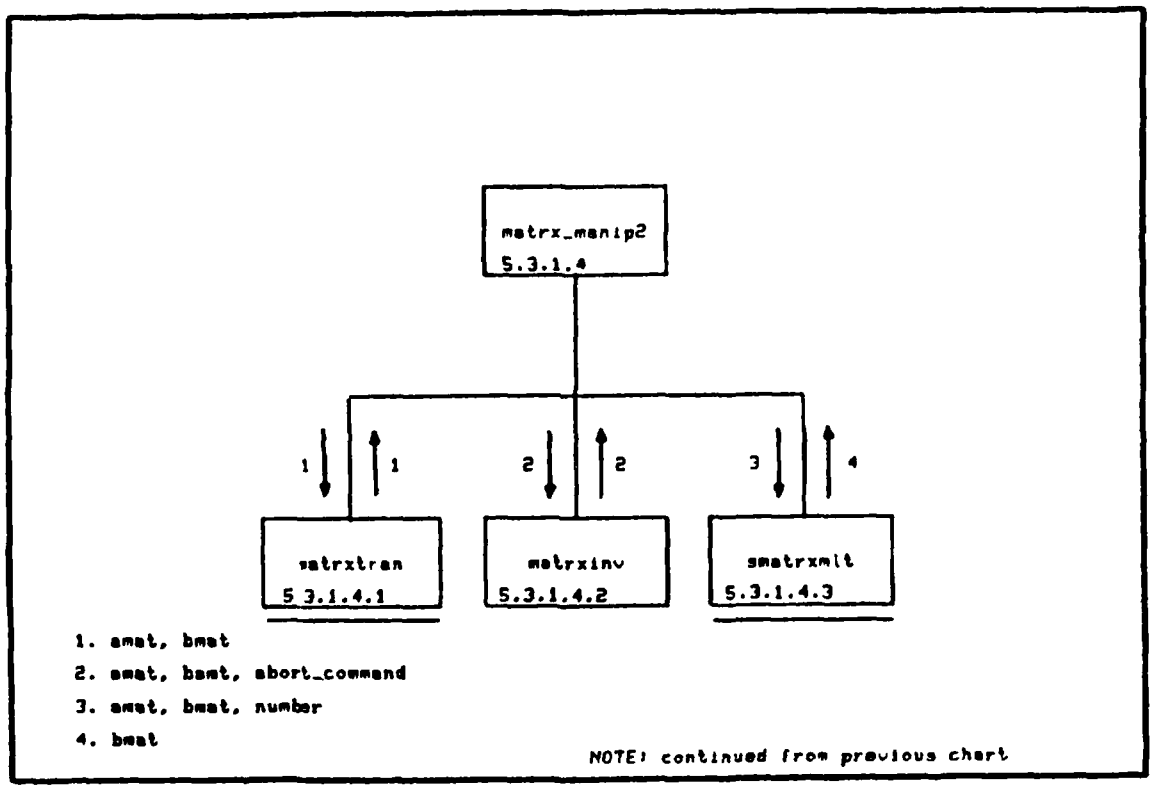
NOTE: Continued from previous page

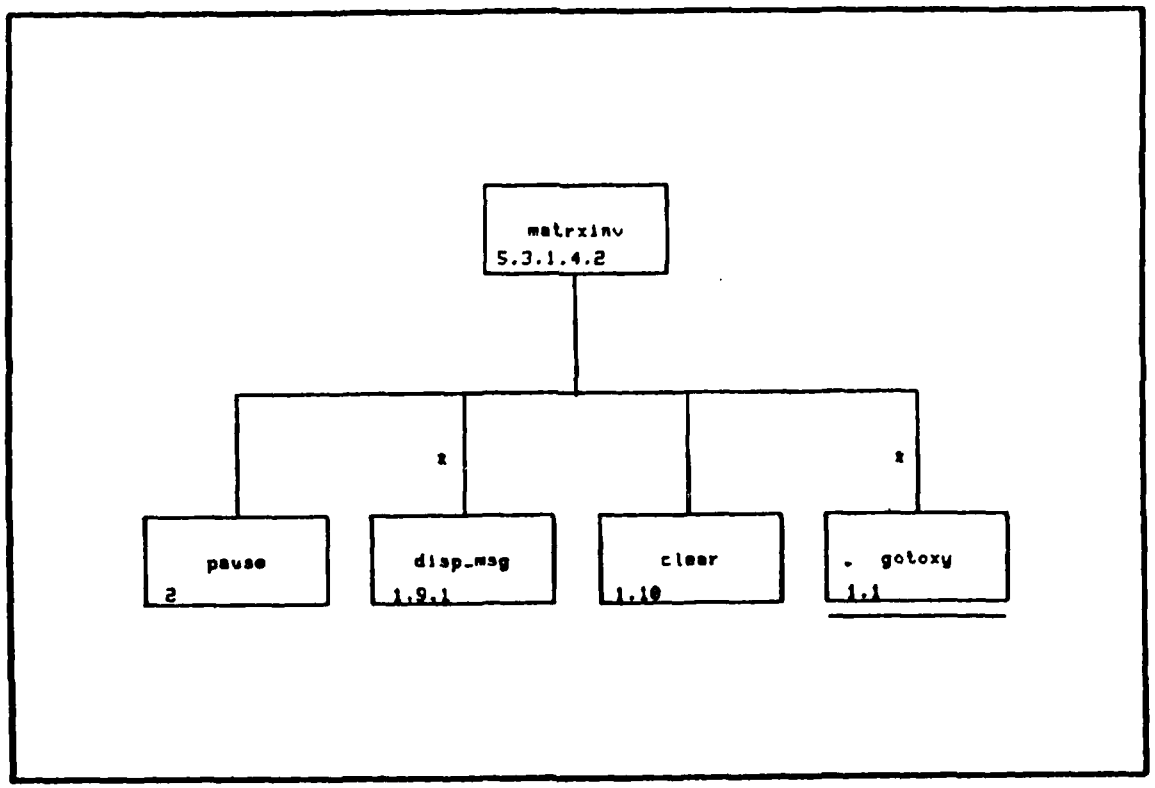

matrxadd
5.3.1.3.1
x
x
clear
1.10
disp_msg
1.9.1
pause
2
gotoxy
1.1

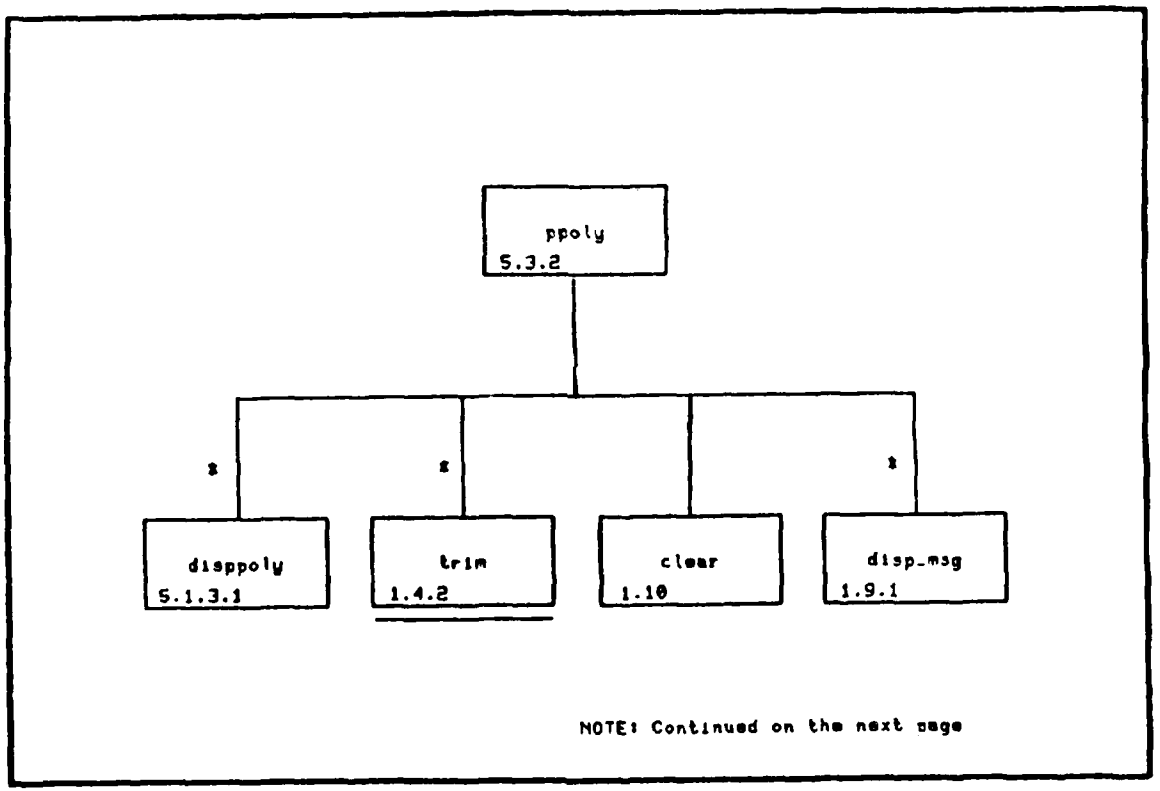

mmtrxmit
5.3.1.3.2
clear
1.10
pause
2
disp_msg
1.9.1
gotoxy
1.1
1
1

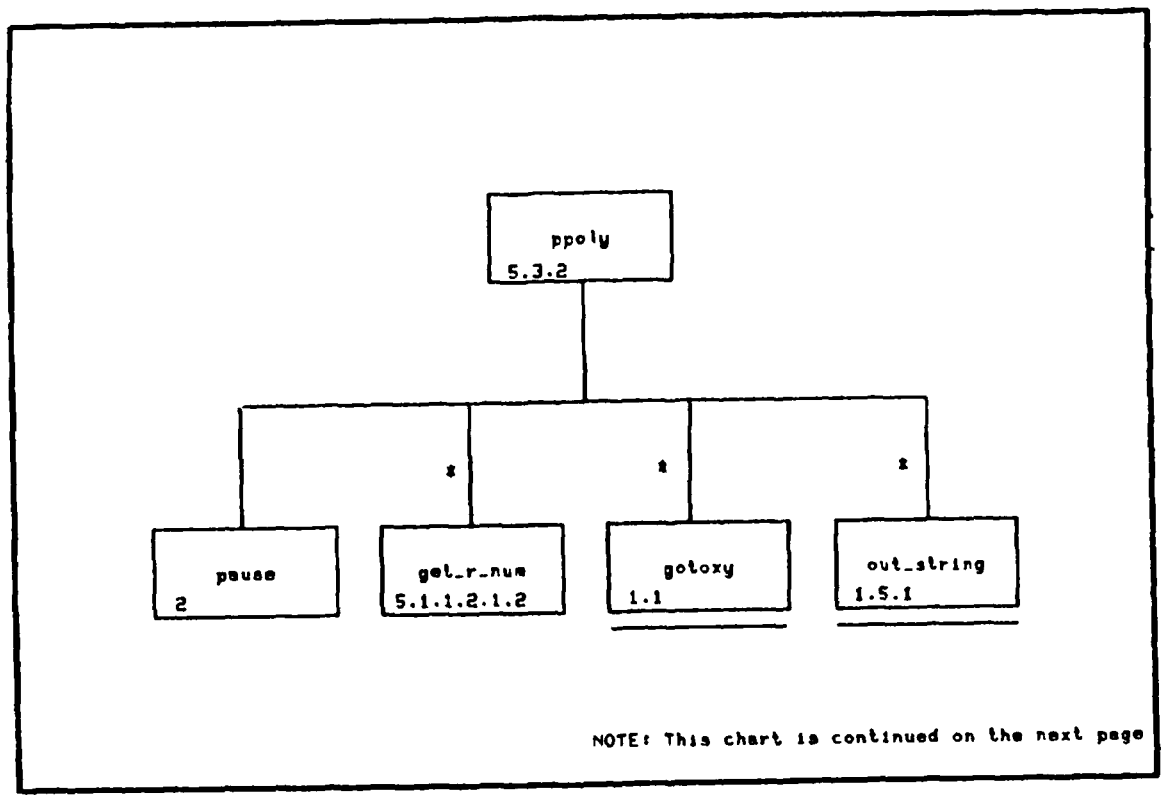

matrxsub
5.3.1.3.3
x
matrxadd
5.3.1.3.1

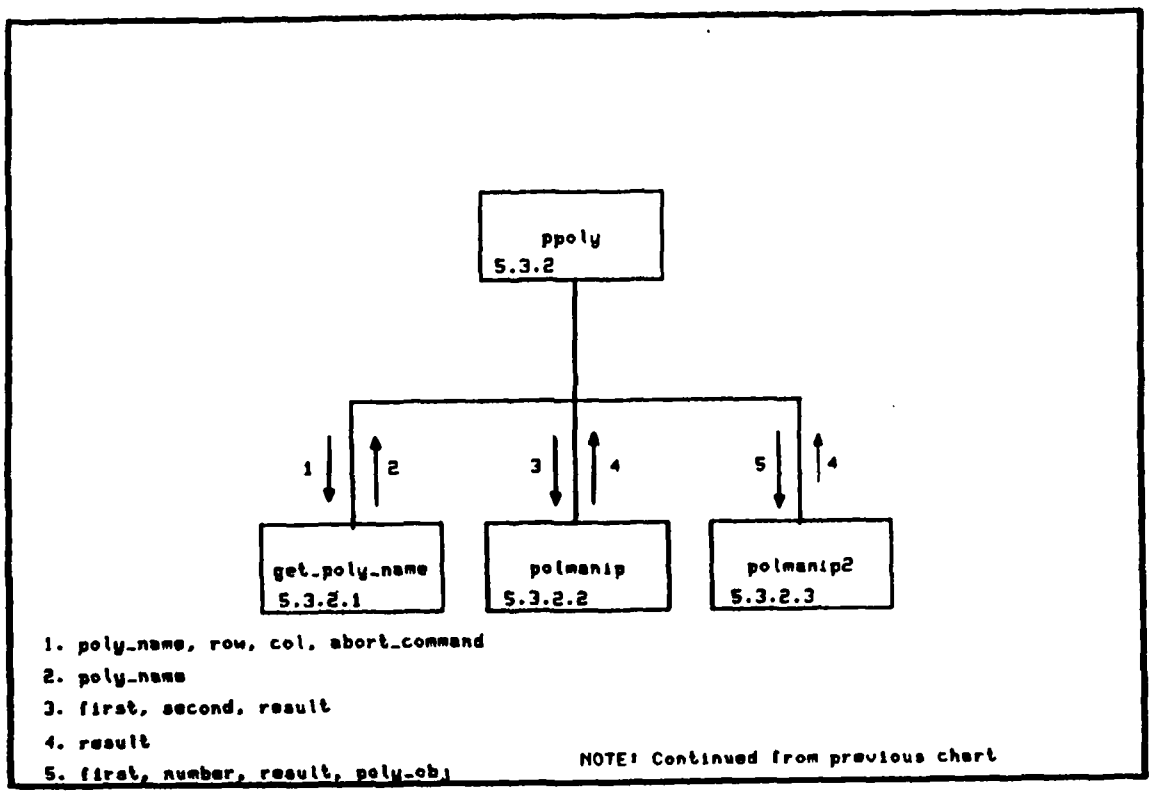

matrx_manip2
5.3.1.4
*
*
disp_matrx
5.3.1.1
trim
1.4.2
pause
2
NOTE: This chart is continued on the next page


matrx_manip2
5.3.1.4
1
1
2
2
3
4
matrxtran
5.3.1.4.1
matrxinv
5.3.1.4.2
smatrxmlt
5.3.1.4.3
1. amat, bmat
2. amat, bamt, abort_command
3. amat, bmat, number
4. bmat
NOTE: continued from previous chart


matrxinv
5.3.1.4.2
pause
2
disp_msg
1.9.1
clear
1.10
gotoxy
1.1

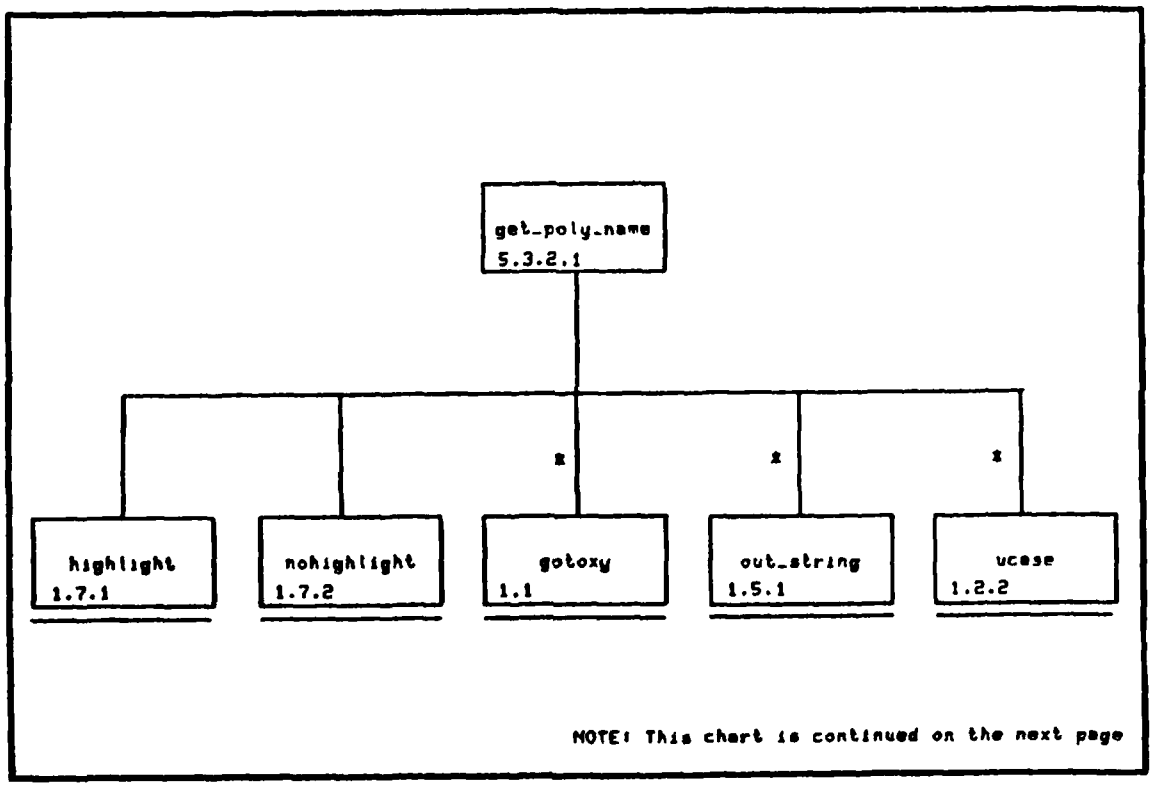

ppoly
5.3.2
x
x
x
disppoly
5.1.3.1
trim
1.4.2
clear
1.10
disp_msg
1.9.1
NOTE: Continued on the next page

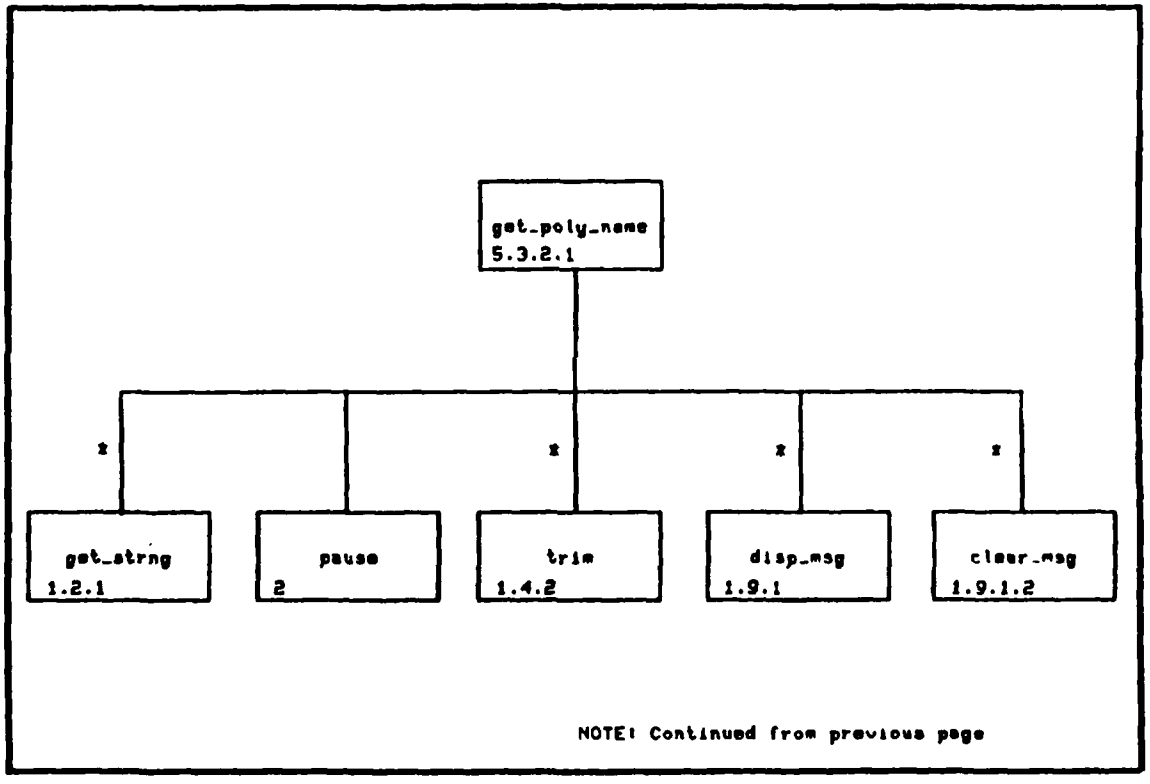

ppoly
5.3.2
x
x
x
pause
2
get_r_num
5.1.1.2.1.2
gotoxy
1.1
out_string
1.5.1
NOTE: This chart is continued on the next page

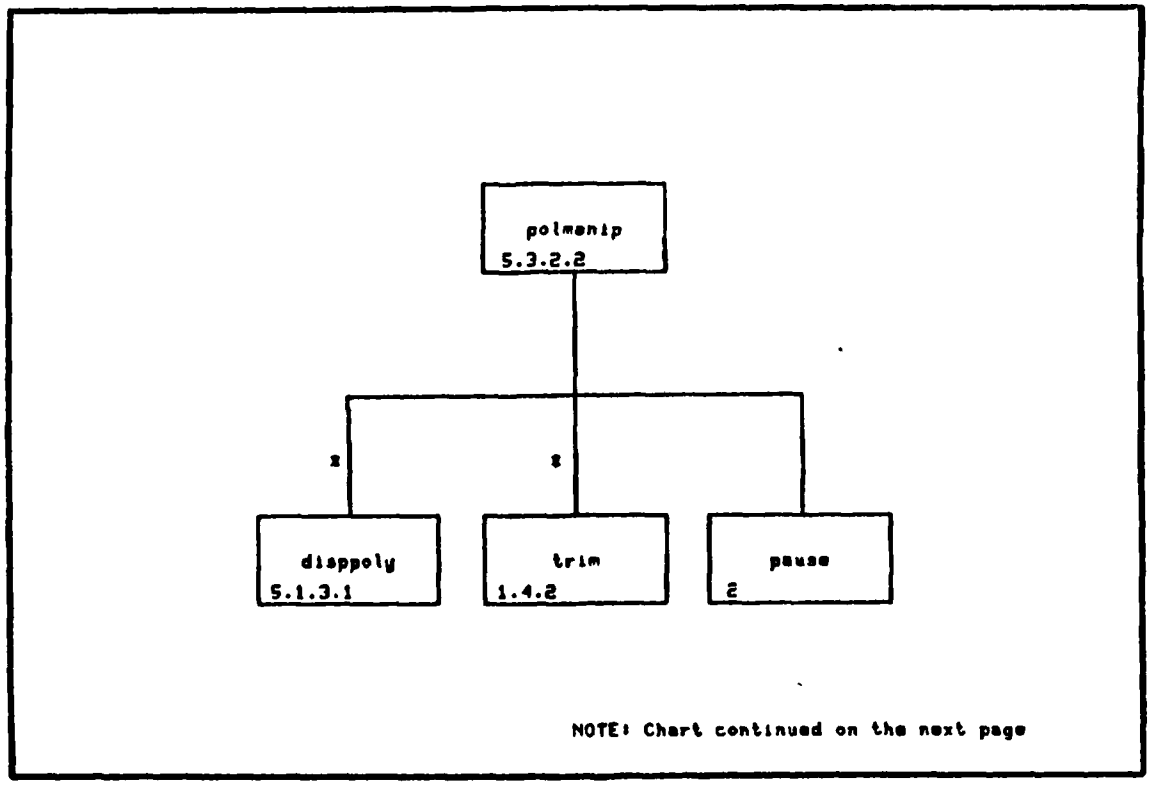


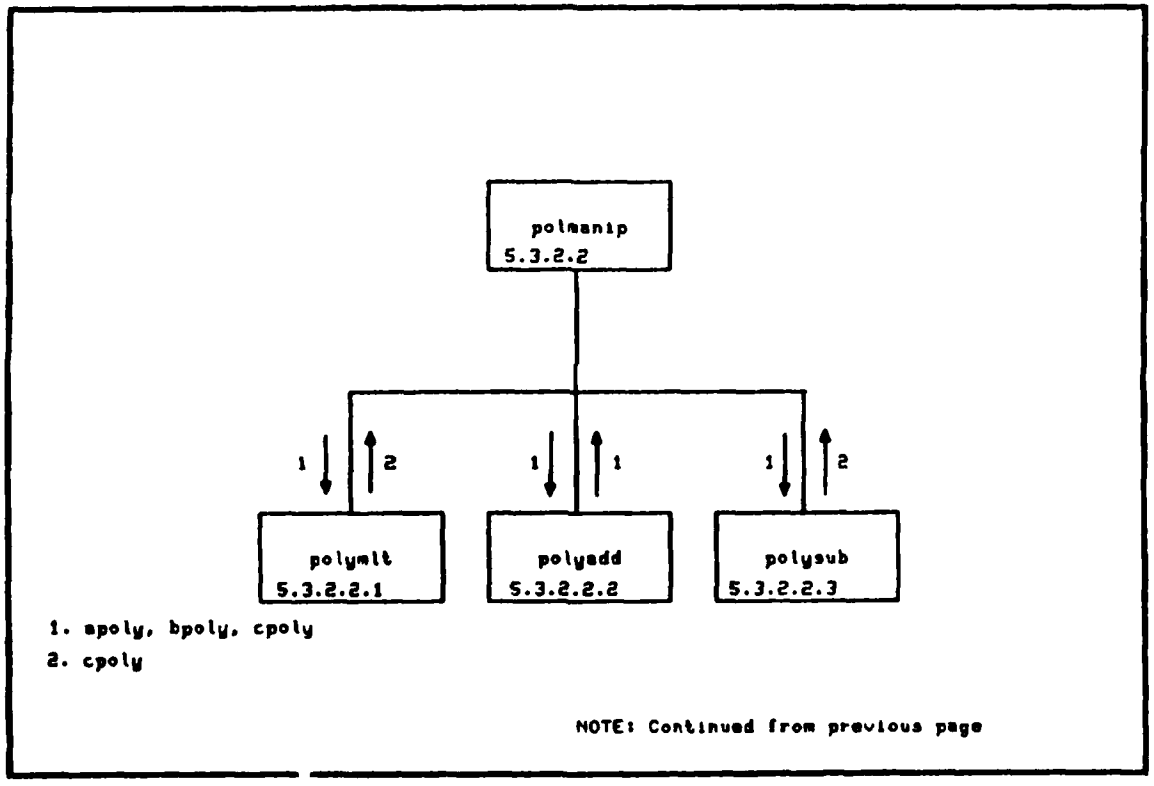
1. poly_name, row, col, abort_command
2. poly_name
3. first, second, result
4. result
5. first, number, result, poly_ob;

NOTE: Continued from previous chart


get_poly_name
5.3.2.1
highlight
1.7.1
nohighlight
1.7.2
gotoxy
1.1
out_string
1.5.1
ucase
1.2.2
NOTE: This chart is continued on the next page

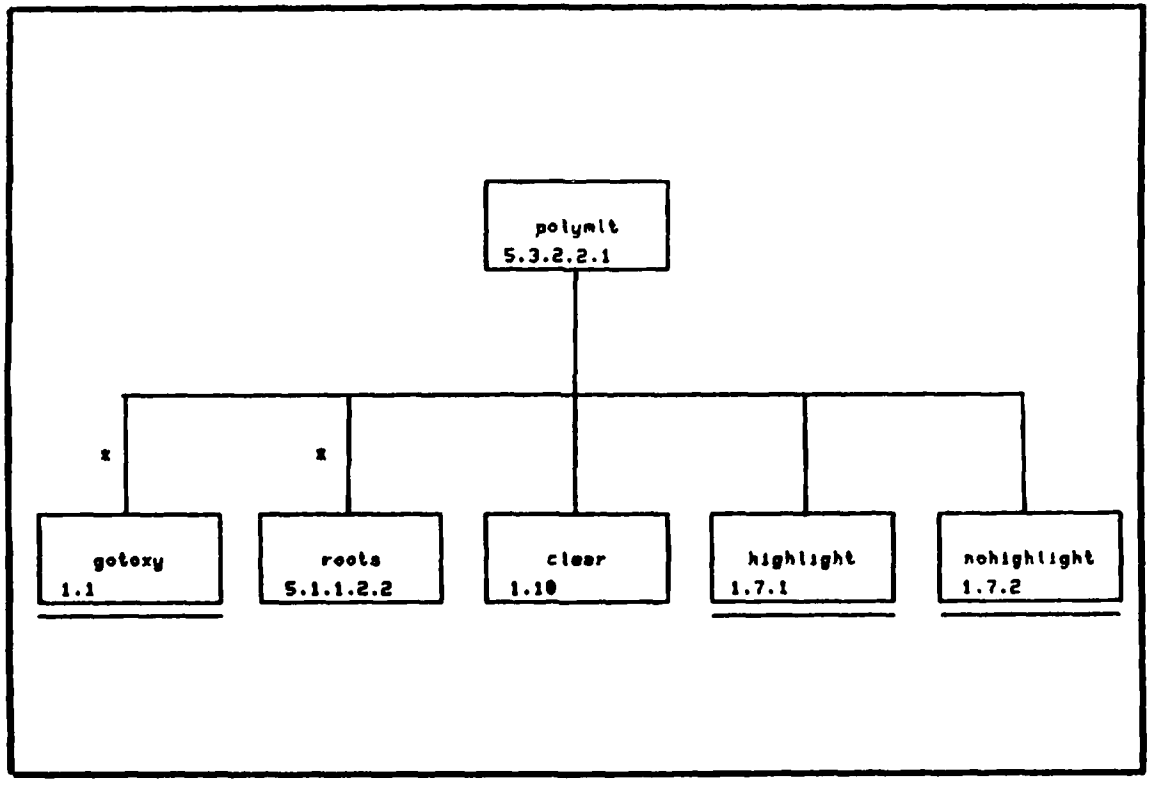

get_poly_name
5.3.2.1
1
1
1
1
get_strng
1.2.1
pause
2
trim
1.4.2
disp_msg
1.9.1
clear_msg
1.9.1.2
NOTE: Continued from previous page

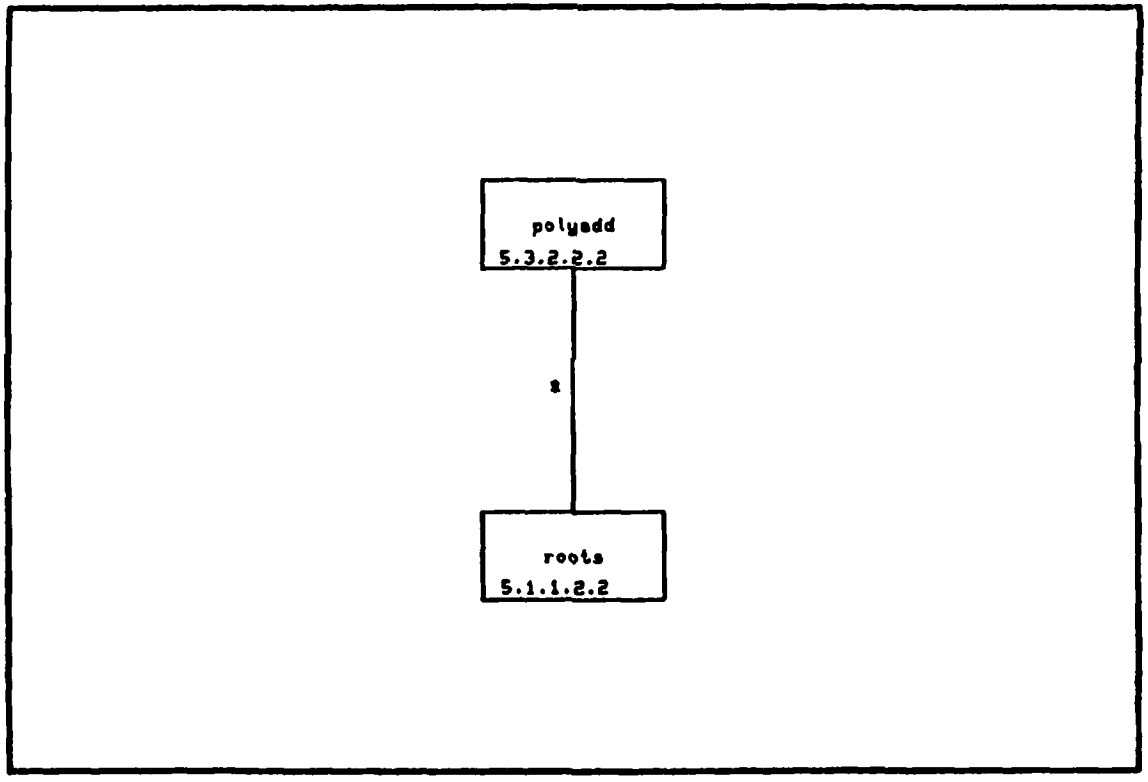

polmanip
5.3.2.2
2
2
disppoly
5.1.3.1
trim
1.4.2
pause
2
NOTE: Chart continued on the next page

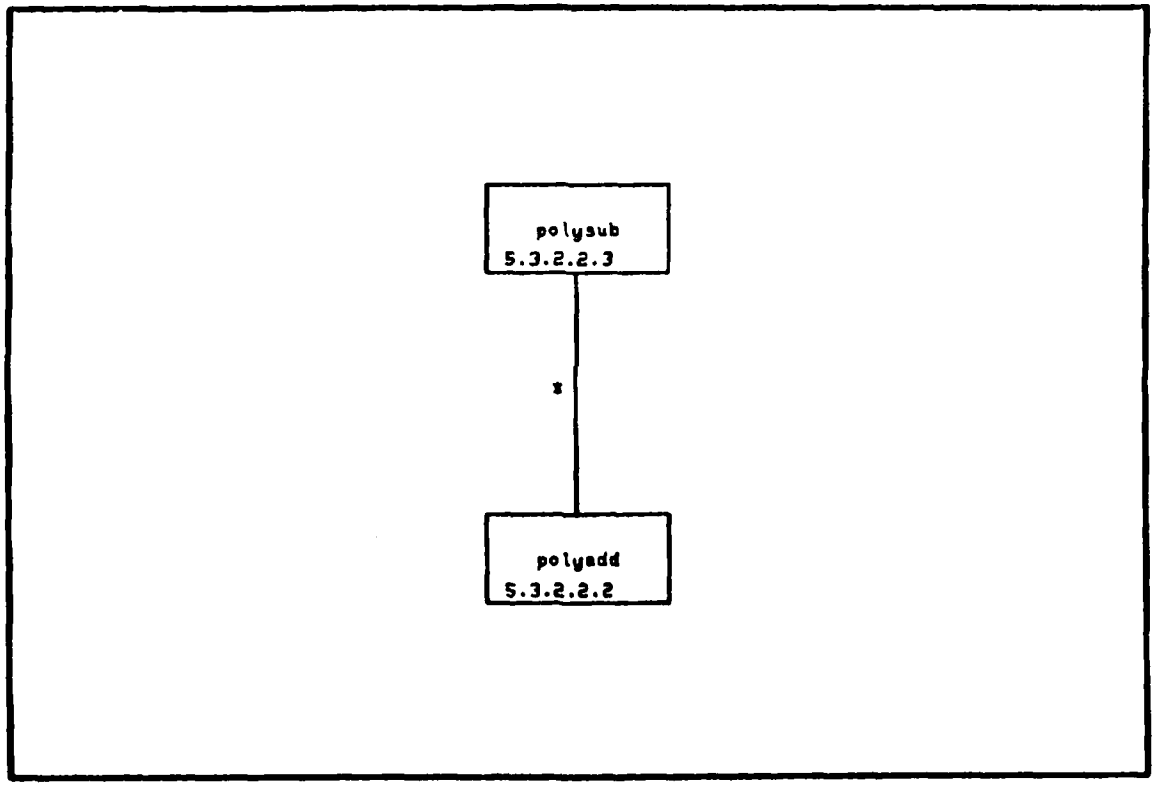


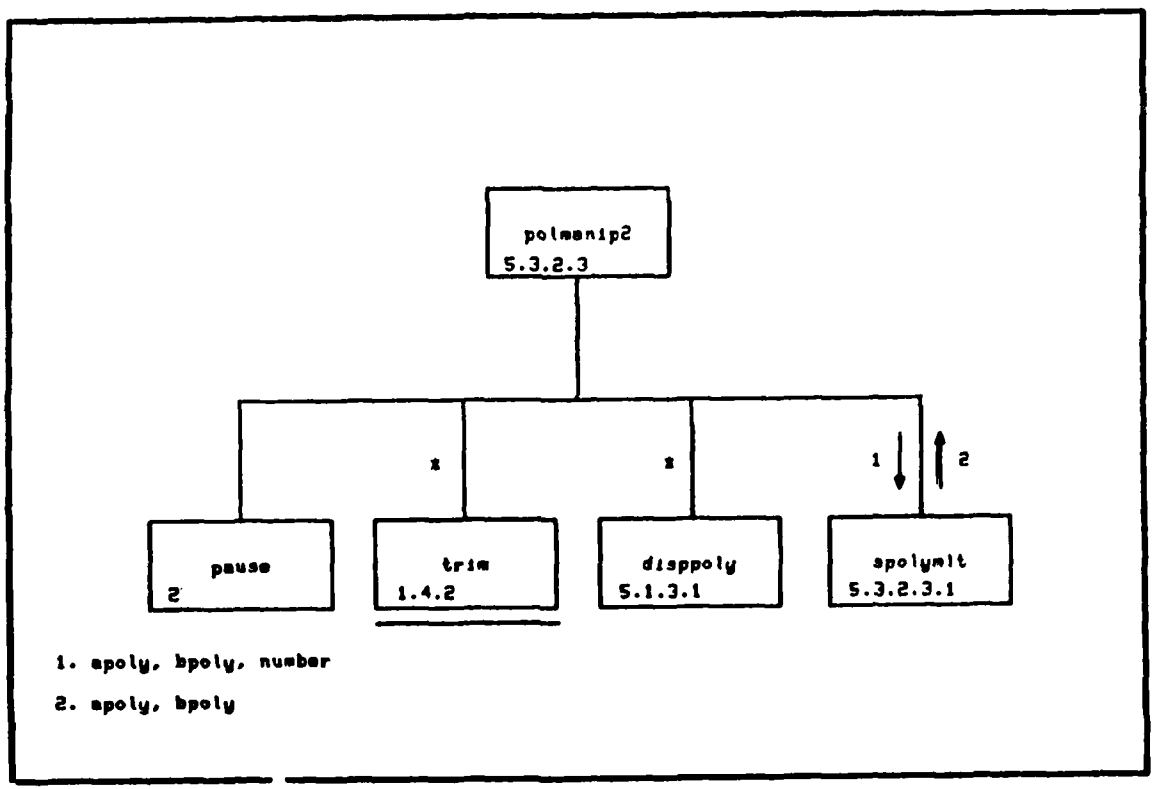
1. apoly, bpoly, cpoly
2. cpoly

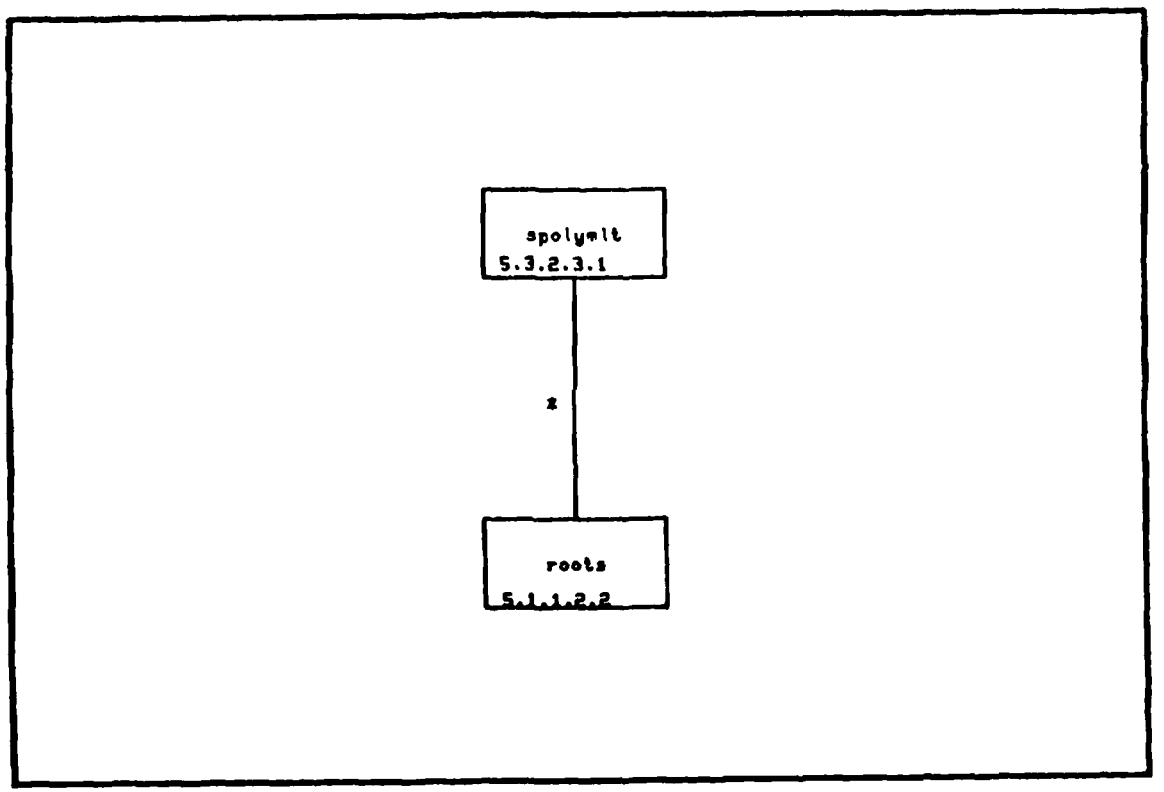
NOTE: Continued from previous page

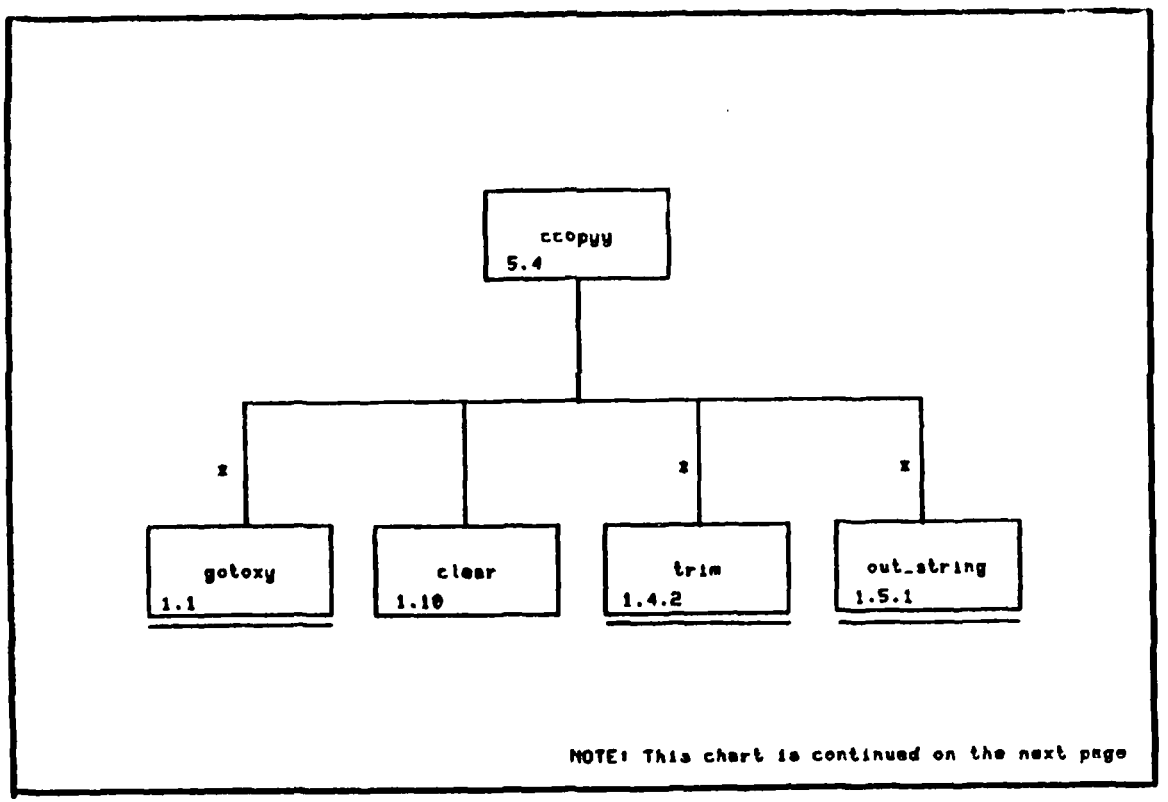

polymlt
5.3.2.2.1
x
x
gotoxy
1.1
roots
5.1.1.2.2
clear
1.10
highlight
1.7.1
nohighlight
1.7.2

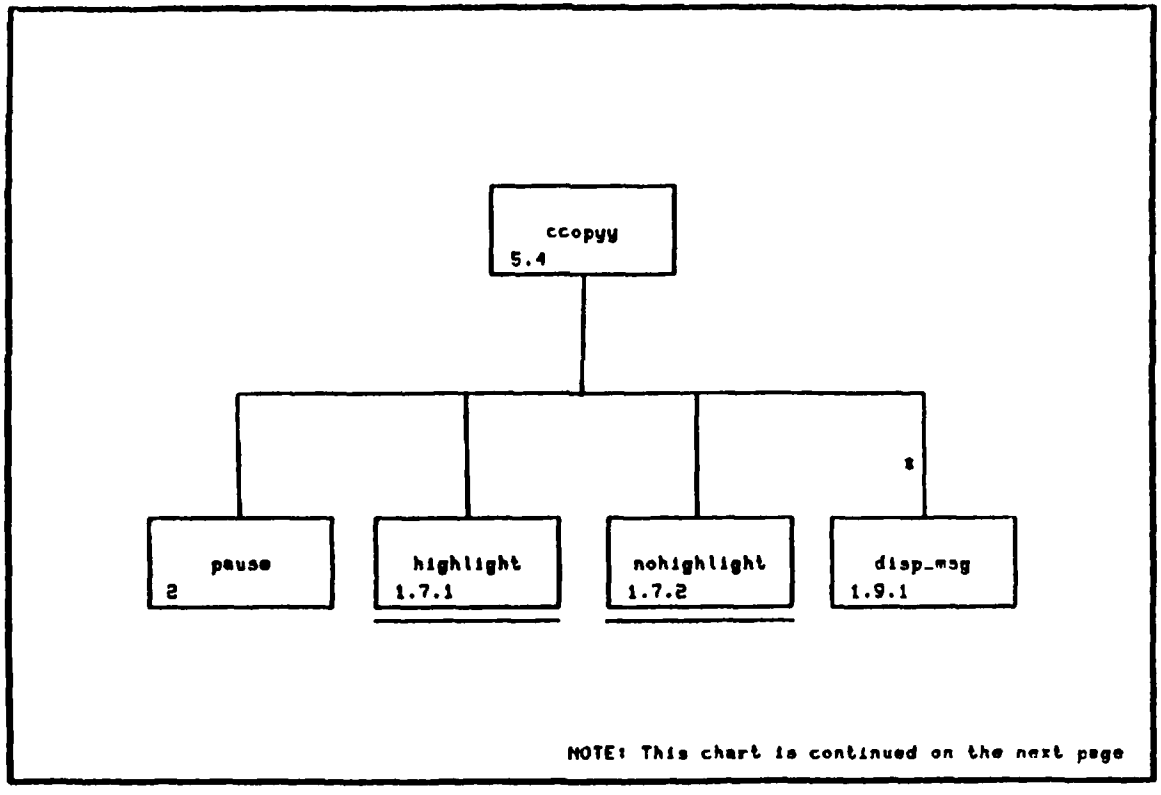

polyadd
5.3.2.2.2
2
roots
5.1.1.2.2

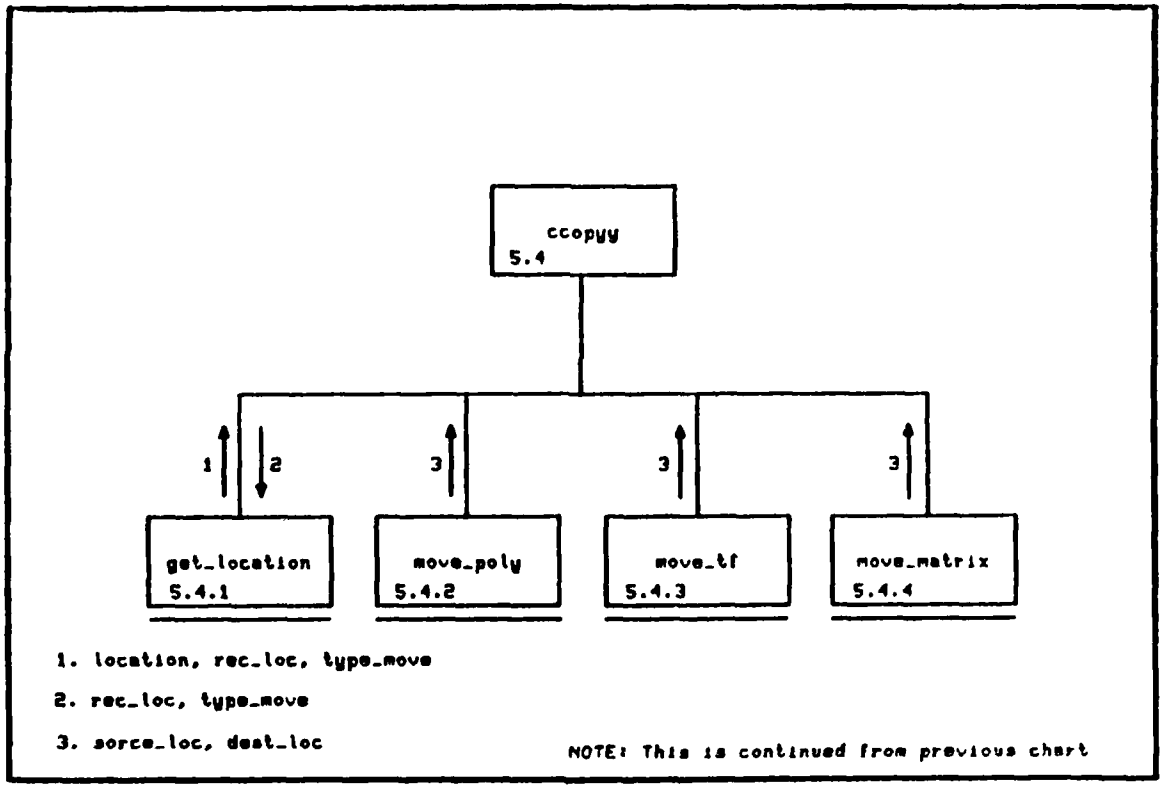

polysub
5.3.2.2.3
x
polyadd
5.3.2.2.2

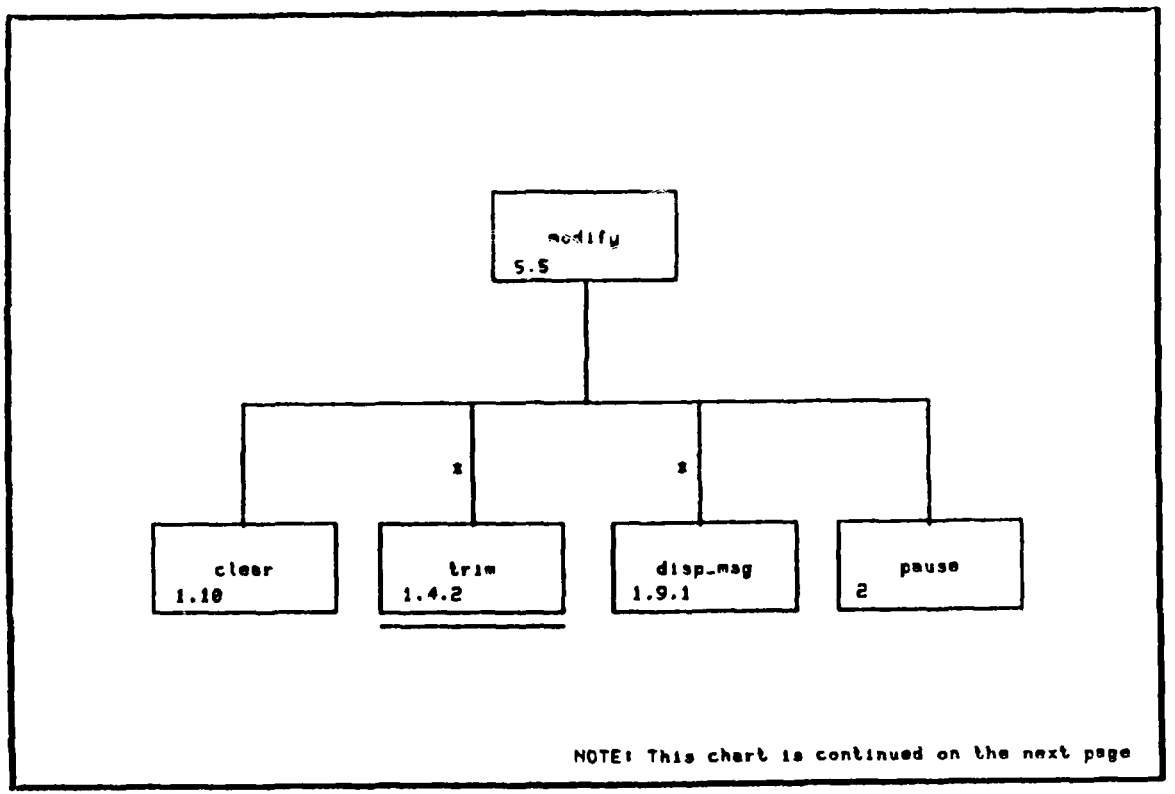

polmanip2
5.3.2.3
pause
2
trim
1.4.2
disppoly
5.1.3.1
spolymlt
5.3.2.3.1
x
x
1
2
1. apoly, bpoly, number
2. apoly, bpoly

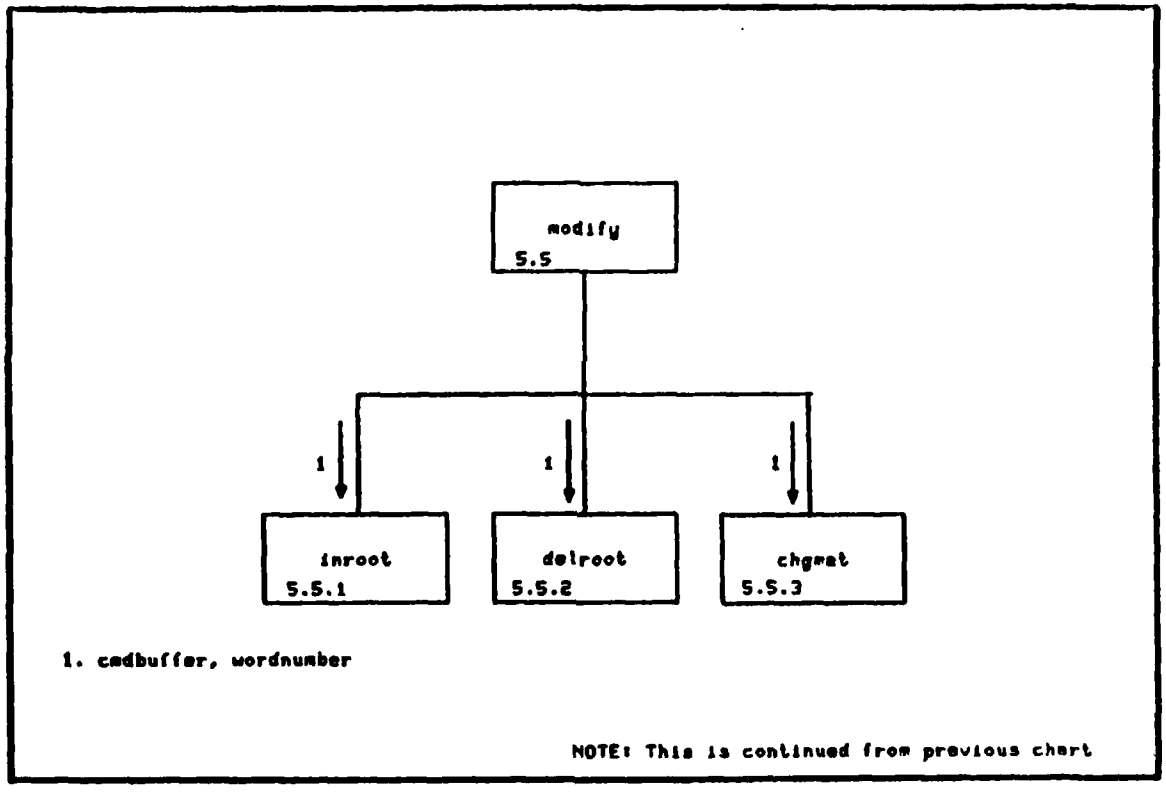

spolymlt
5.3.2.3.1
z
roots
5.1.1.2.2


ccopyy
5.4
x
x
x
gotoxy
1.1
clear
1.10
trim
1.4.2
out_string
1.5.1
NOTE: This chart is continued on the next page


ccopyy
5.4
pause
2
highlight
1.7.1
nohighlight
1.7.2
disp_msg
1.9.1
1
NOTE: This chart is continued on the next page

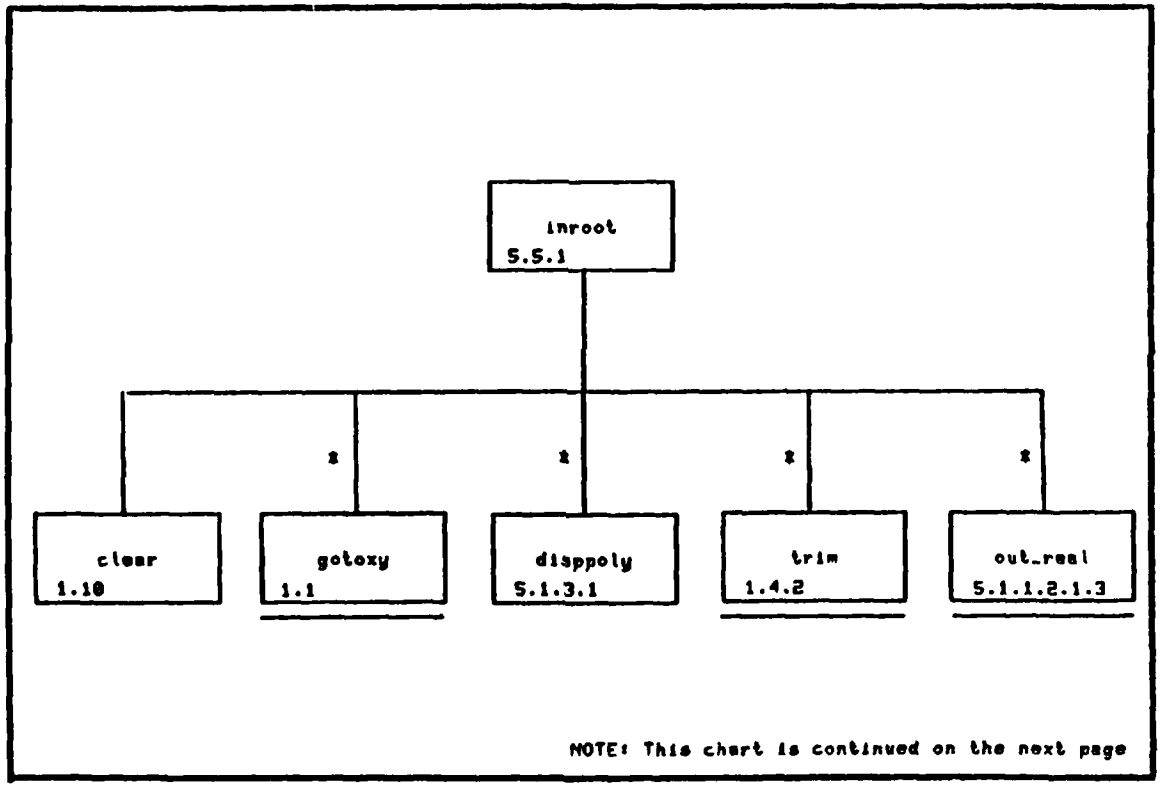

ccopyy
5.4
1
2
3
3
3
get_location
5.4.1
move_poly
5.4.2
move_tf
5.4.3
move_matrix
5.4.4
1. location, rec_loc, type_move
2. rec_loc, type_move
3. sorce_loc, dest_loc
NOTE: This is continued from previous chart

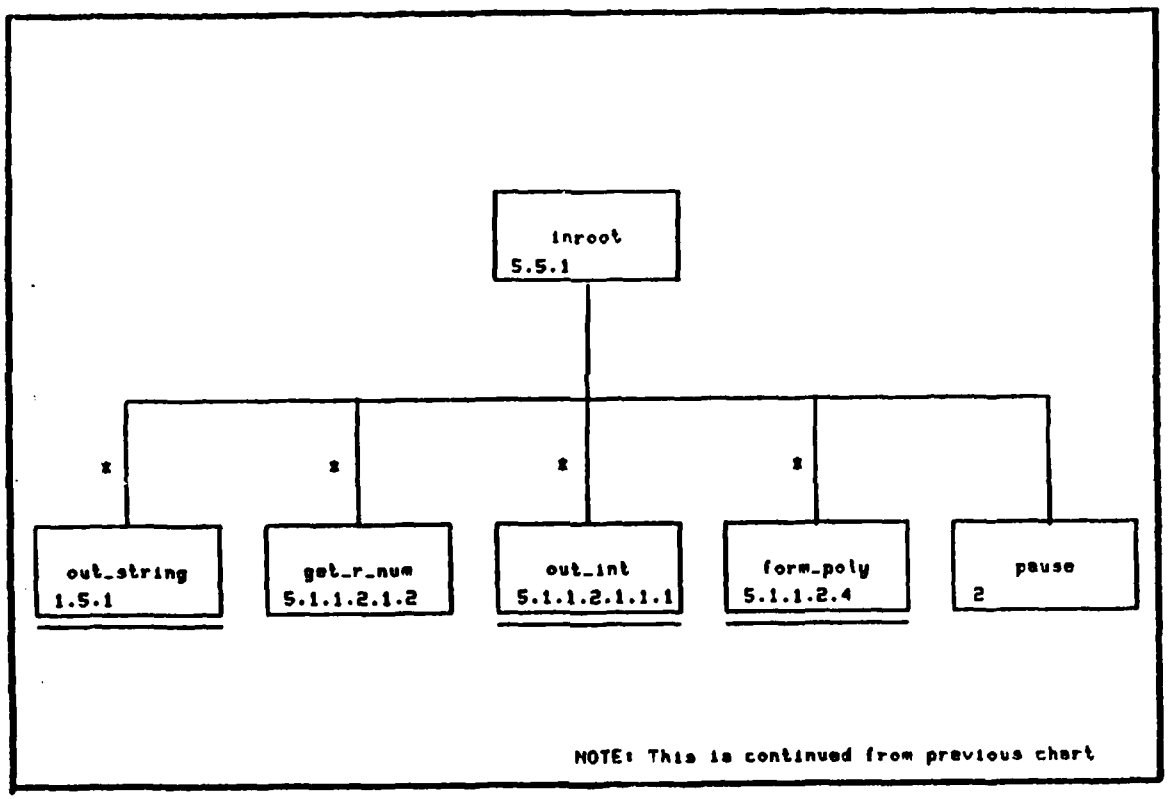

modify
5.5
x
x
clear
1.10
trim
1.4.2
disp_msg
1.9.1
pause
2
NOTE: This chart is continued on the next page

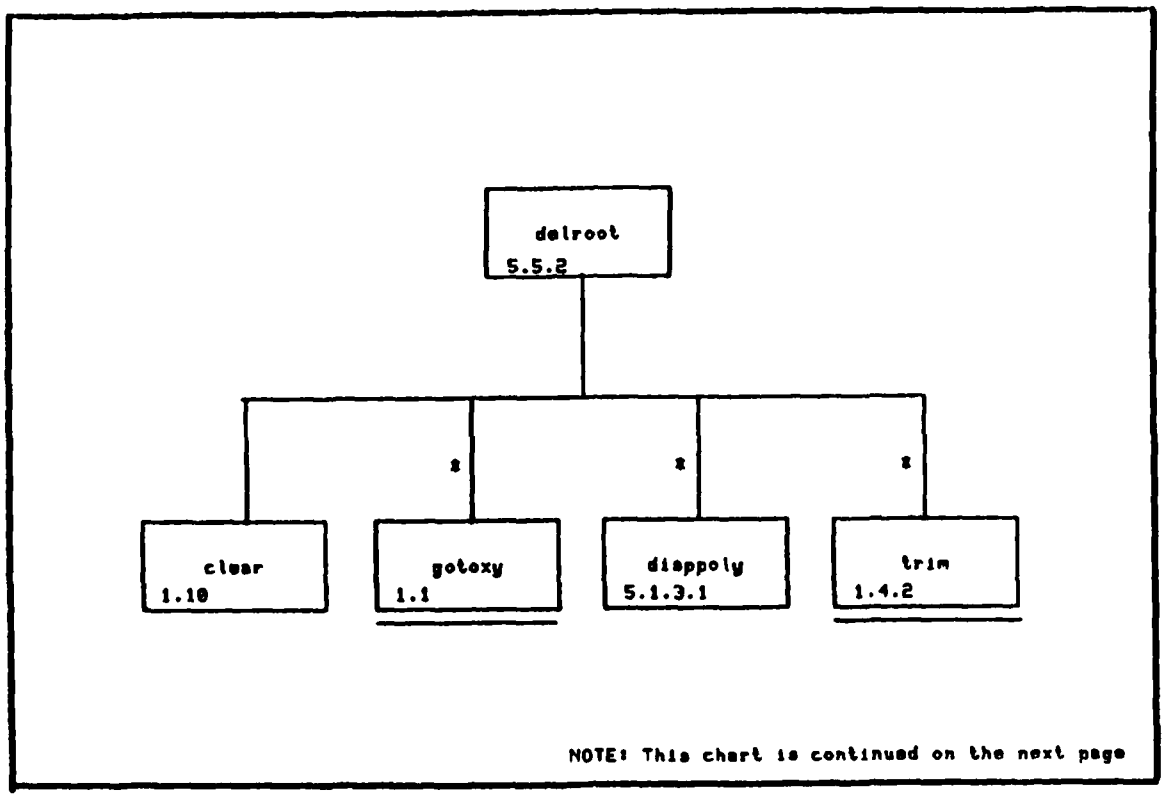


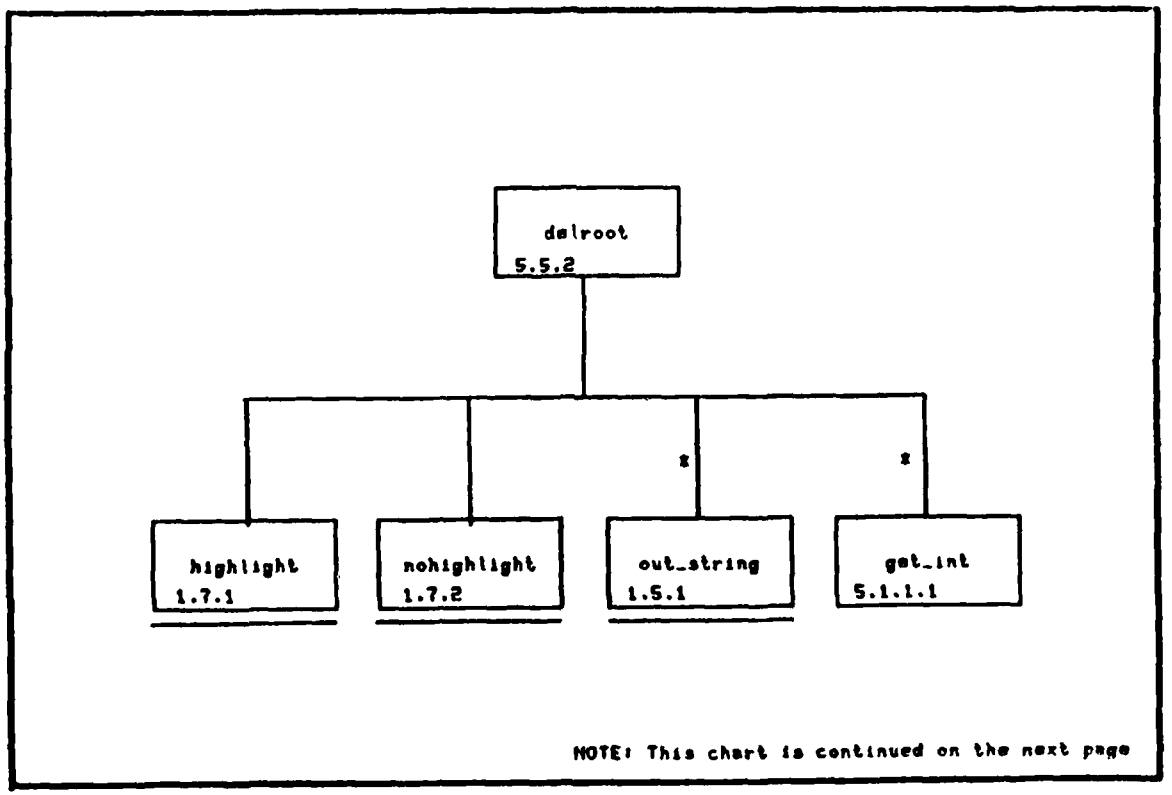
1. cmdbuffer, wordnumber

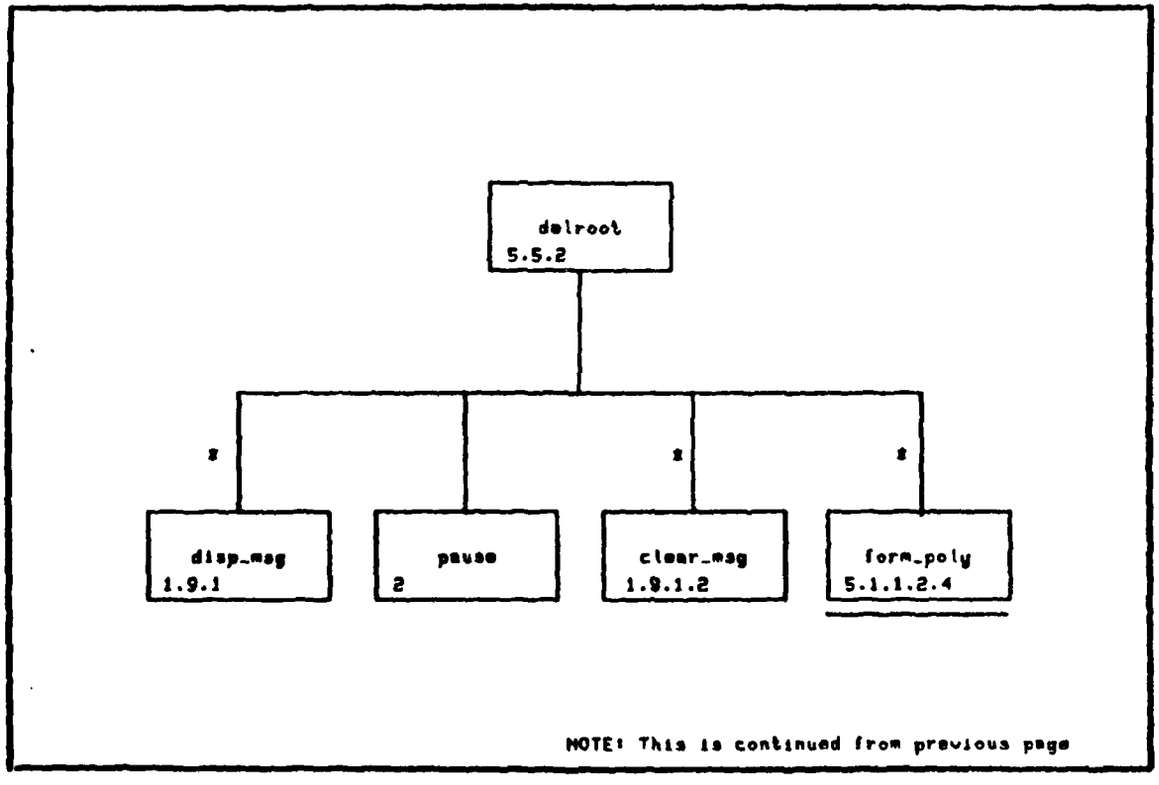
NOTE: This is continued from previous chart

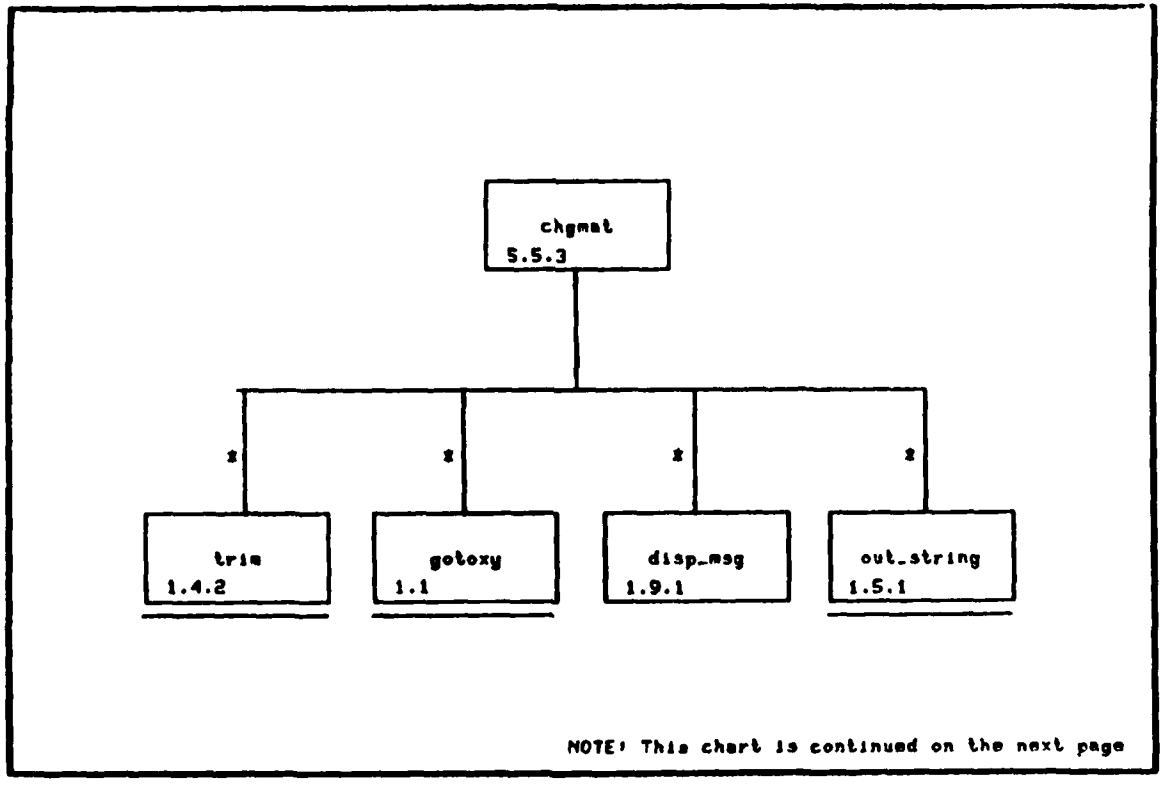

inroot
5.5.1
clear
1.10
gotoxy
1.1
disppoly
5.1.3.1
trim
1.4.2
out_real
5.1.1.2.1.3
NOTE: This chart is continued on the next page


inroot
5.5.1
x
x
x
x
out_string
1.5.1
get_r_num
5.1.1.2.1.2
out_int
5.1.1.2.1.1.1
form_poly
5.1.1.2.4
pause
2
NOTE: This is continued from previous chart

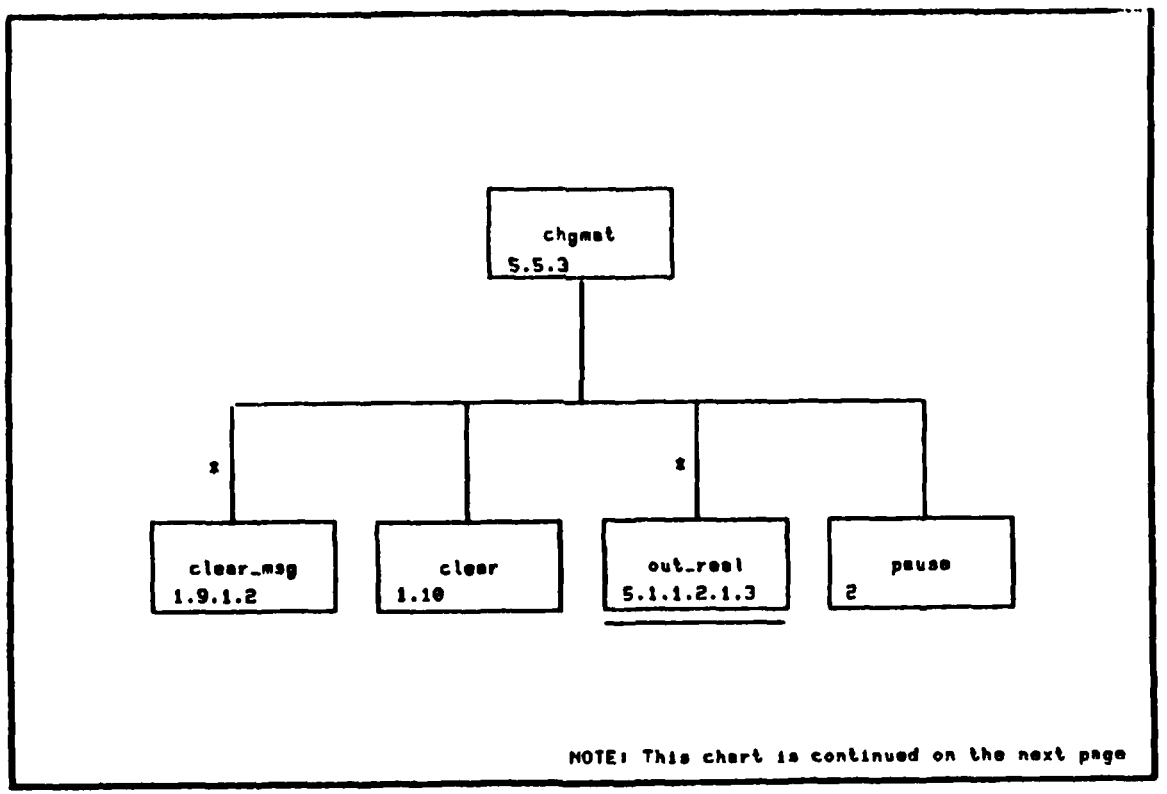

delroot
5.5.2
clear
1.10
gotoxy
1.1
dispposly
5.1.3.1
trim
1.4.2
NOTE: This chart is continued on the next page

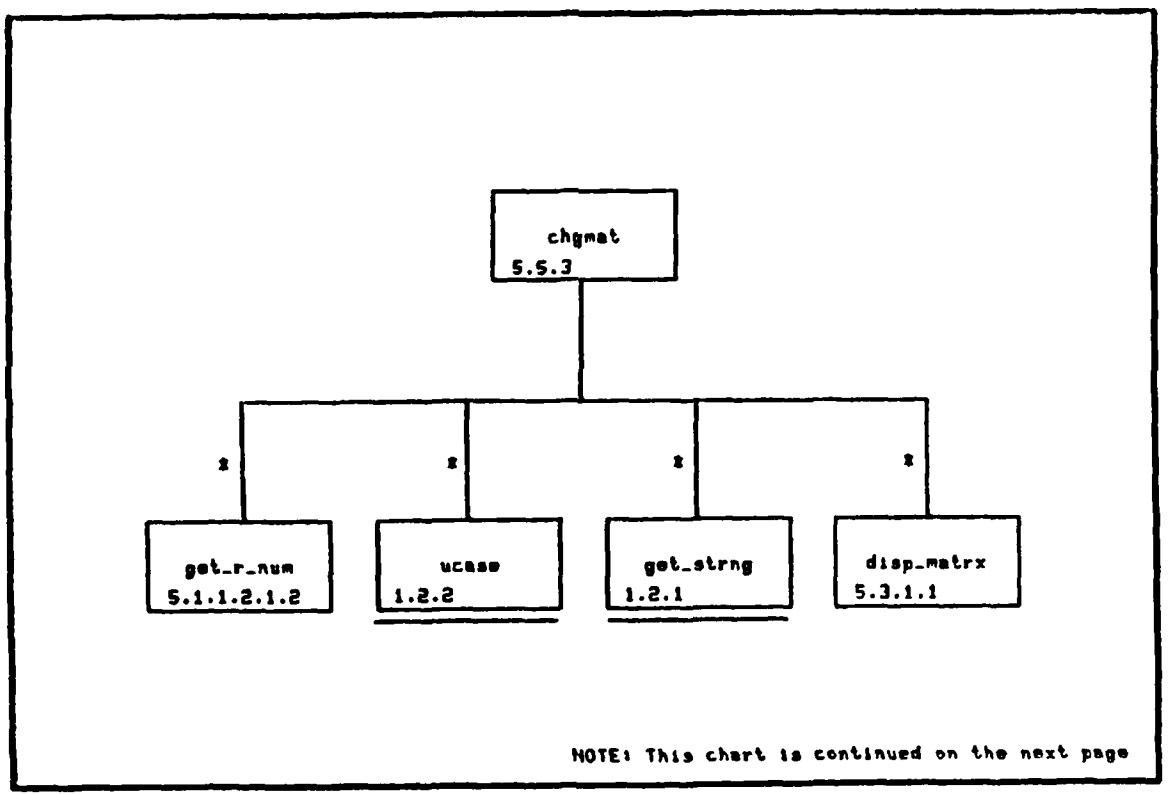

delroot
5.5.2
highlight
1.7.1
nohighlight
1.7.2
out_string
1.5.1
get_int
5.1.1.1
NOTE: This chart is continued on the next page

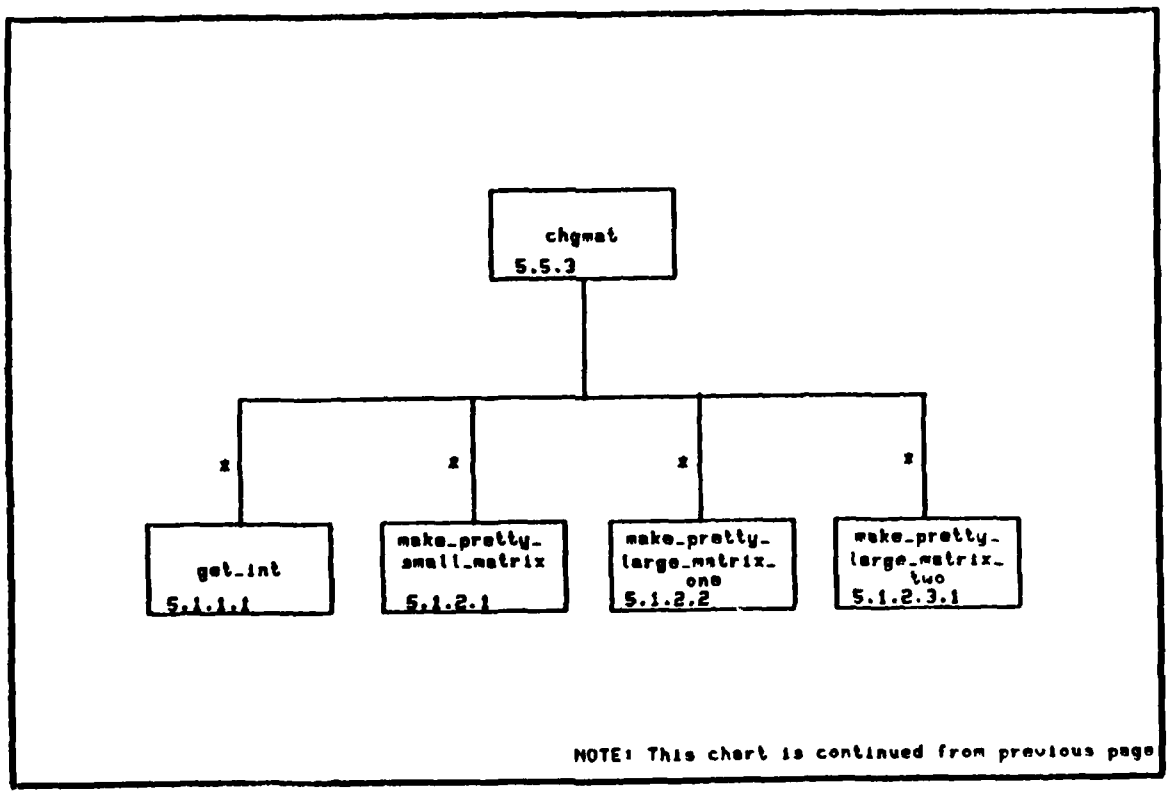

delroot
5.5.2
1
1
1
disp_msg
1.9.1
pause
2
clear_msg
1.8.1.2
form_poly
5.1.1.2.4
NOTE: This is continued from previous page

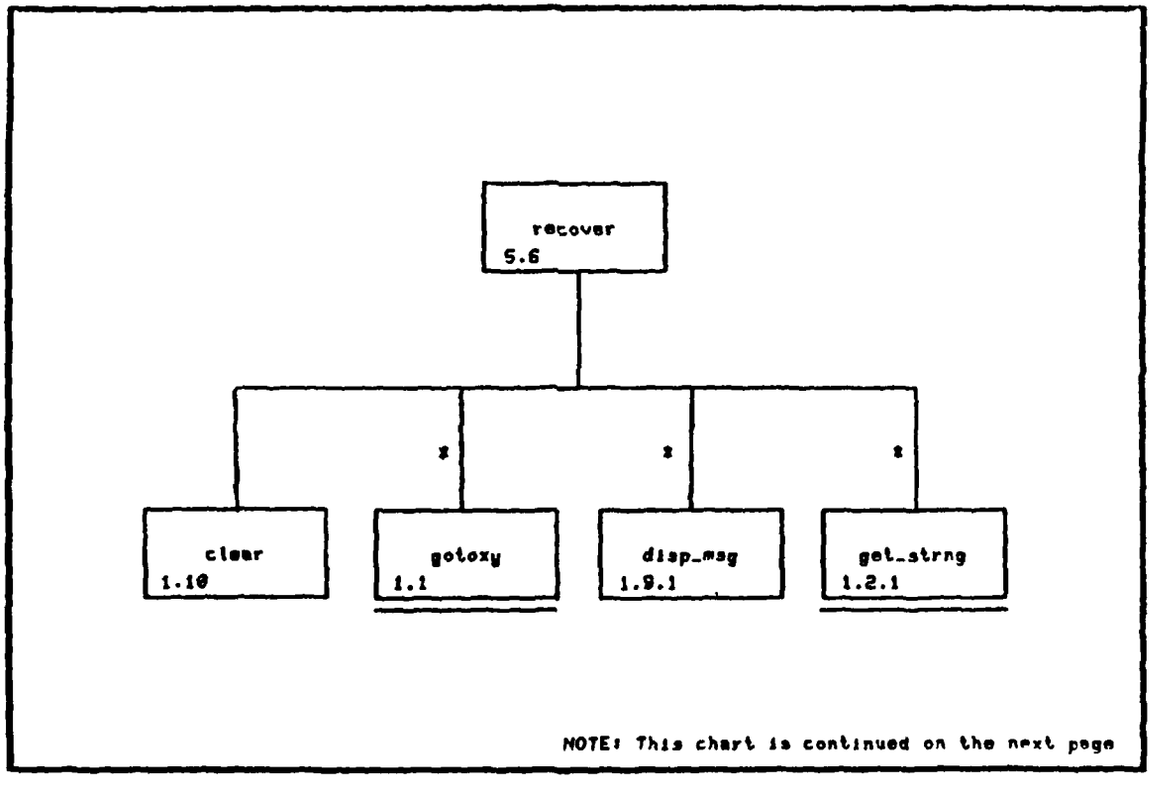


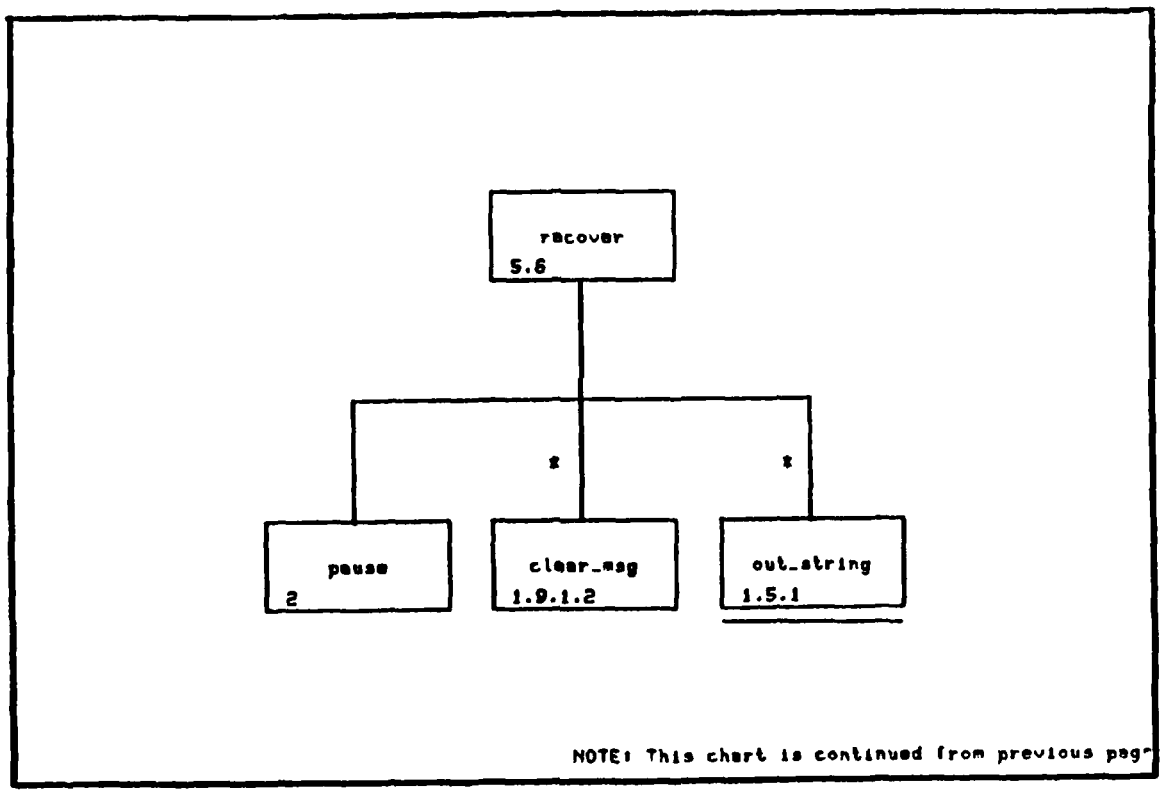
NOTE: This chart is continued on the next page

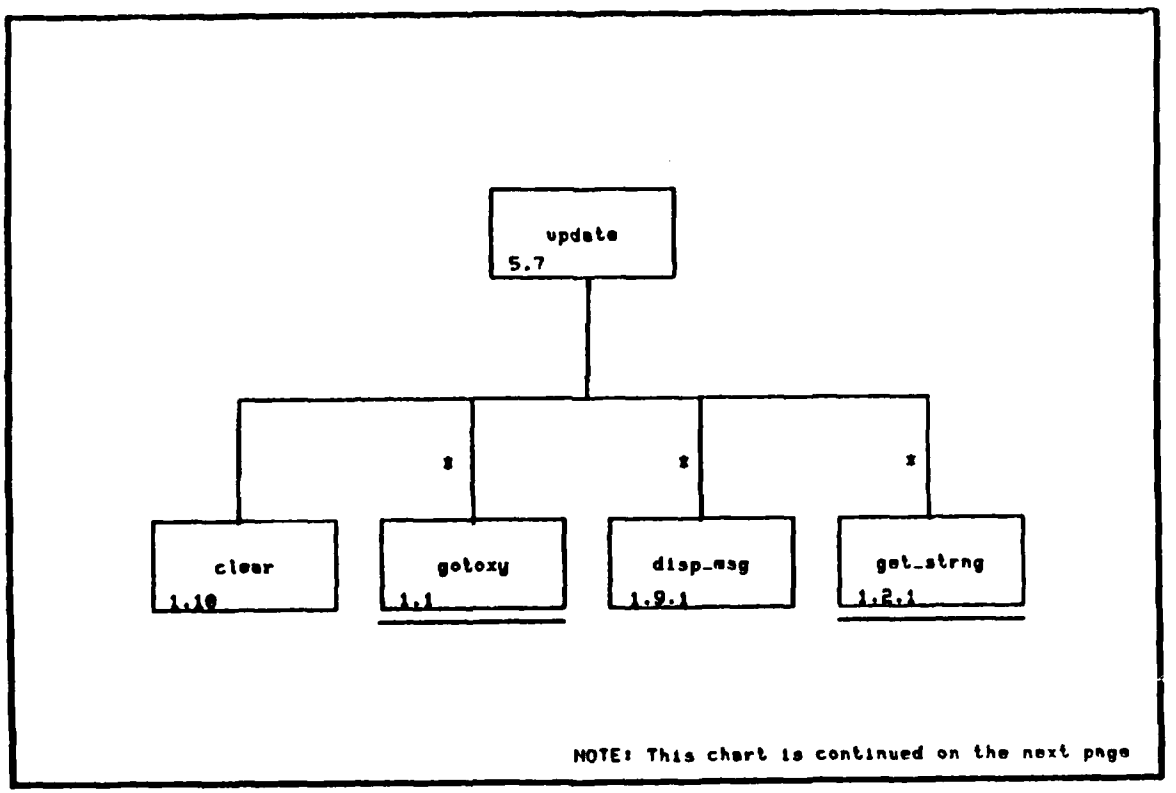

chgmat
5.5.3
2
2
clear_msg
1.9.1.2
clear
1.10
out_real
5.1.1.2.1.3
pause
2
NOTE: This chart is continued on the next page

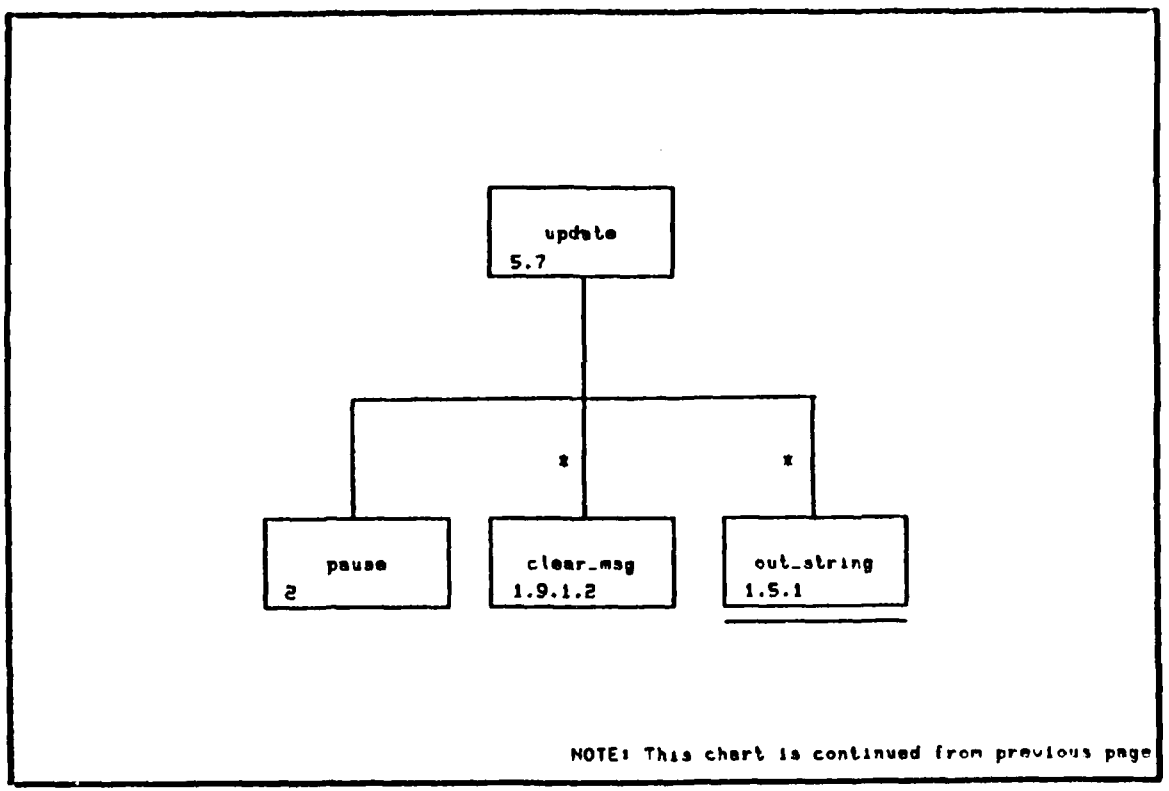

chgmat
5.5.3
get_r_num
5.1.1.2.1.2
ucase
1.2.2
get_strng
1.2.1
disp_matrx
5.3.1.1
NOTE: This chart is continued on the next page

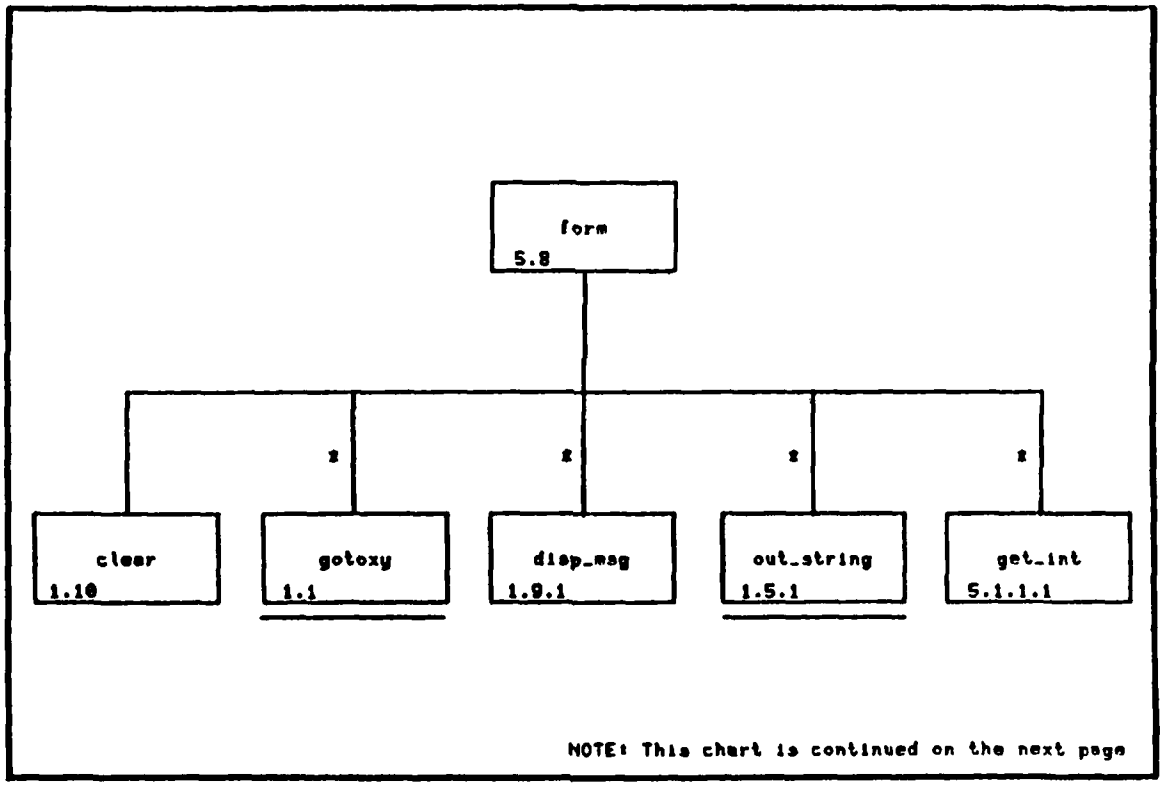

chgmat
5.5.3
2
2
2
2
get_int
5.1.1.1
make_pretty_
small_matrix
5.1.2.1
make_pretty_
large_matrix_
one
5.1.2.2
make_pretty_
large_matrix_
two
5.1.2.3.1
NOTE: This chart is continued from previous page

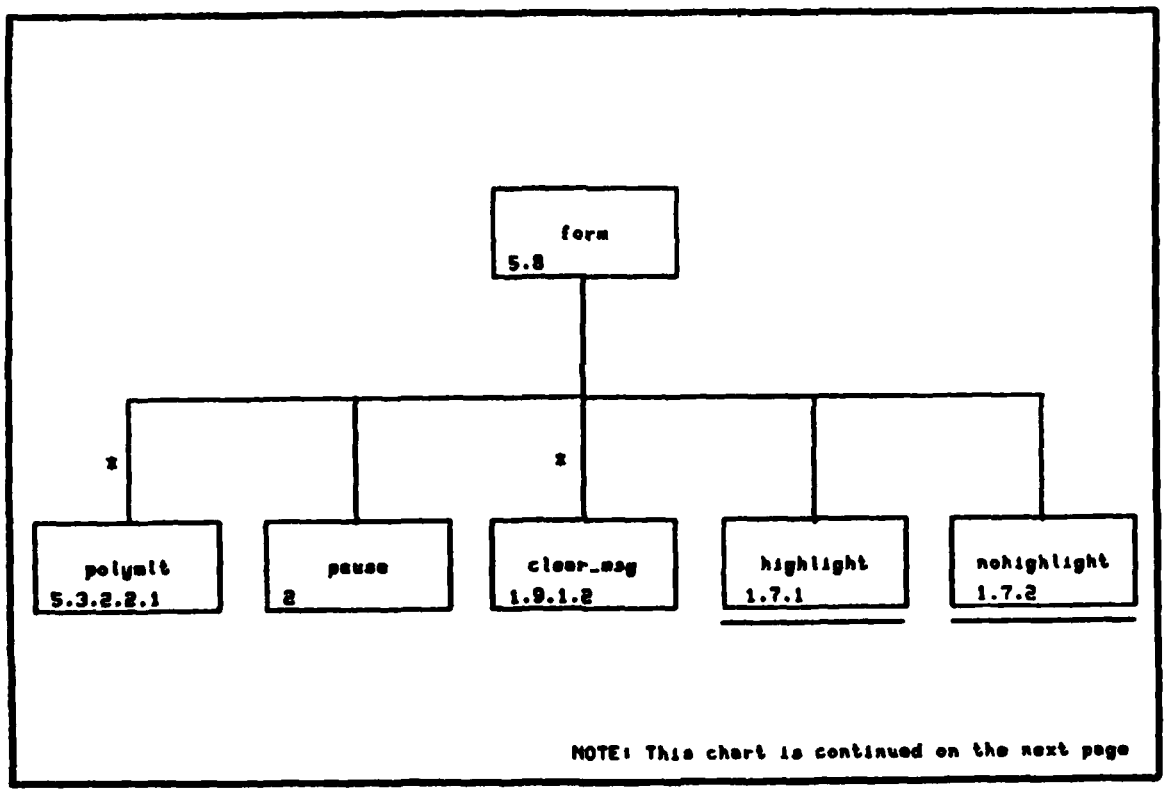

recover
5.6
x
x
x
clear
1.10
gotoxy
1.1
disp_msg
1.9.1
get_strng
1.2.1
NOTE: This chart is continued on the next page

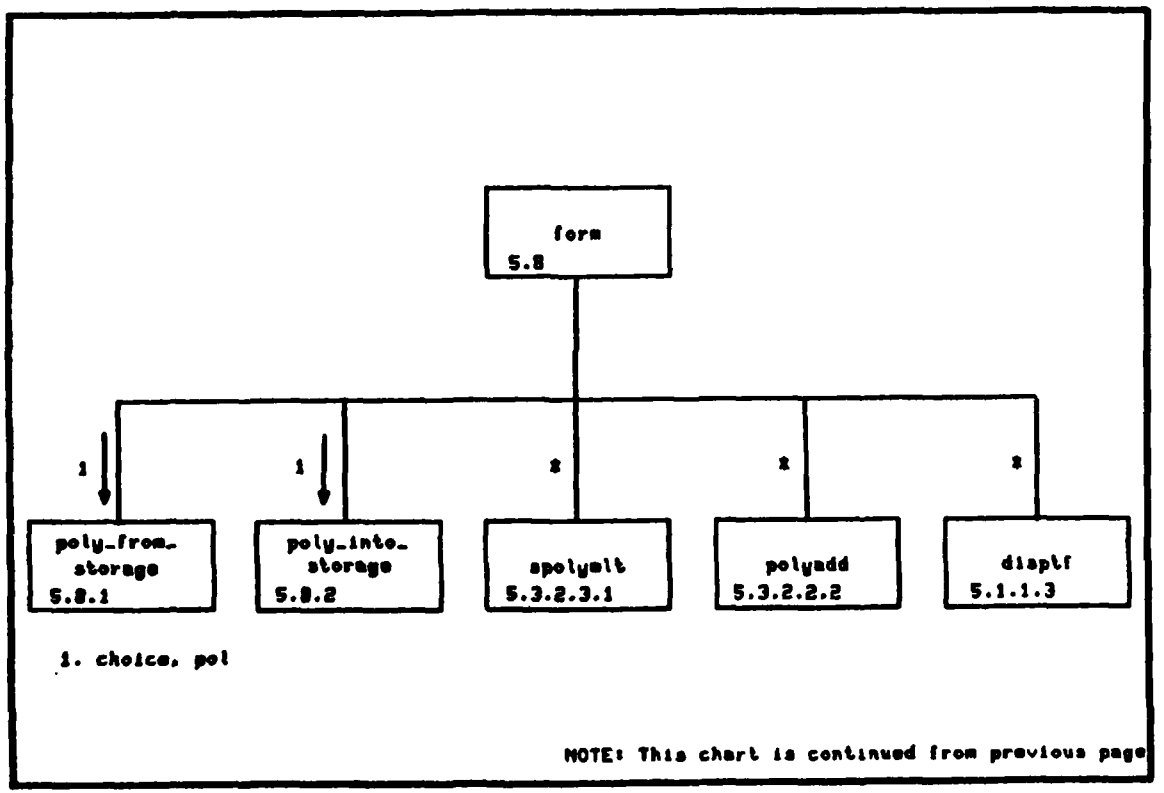

recover
5.6
2
2
pause
2
clear_msg
1.9.1.2
out_string
1.5.1
NOTE: This chart is continued from previous pag

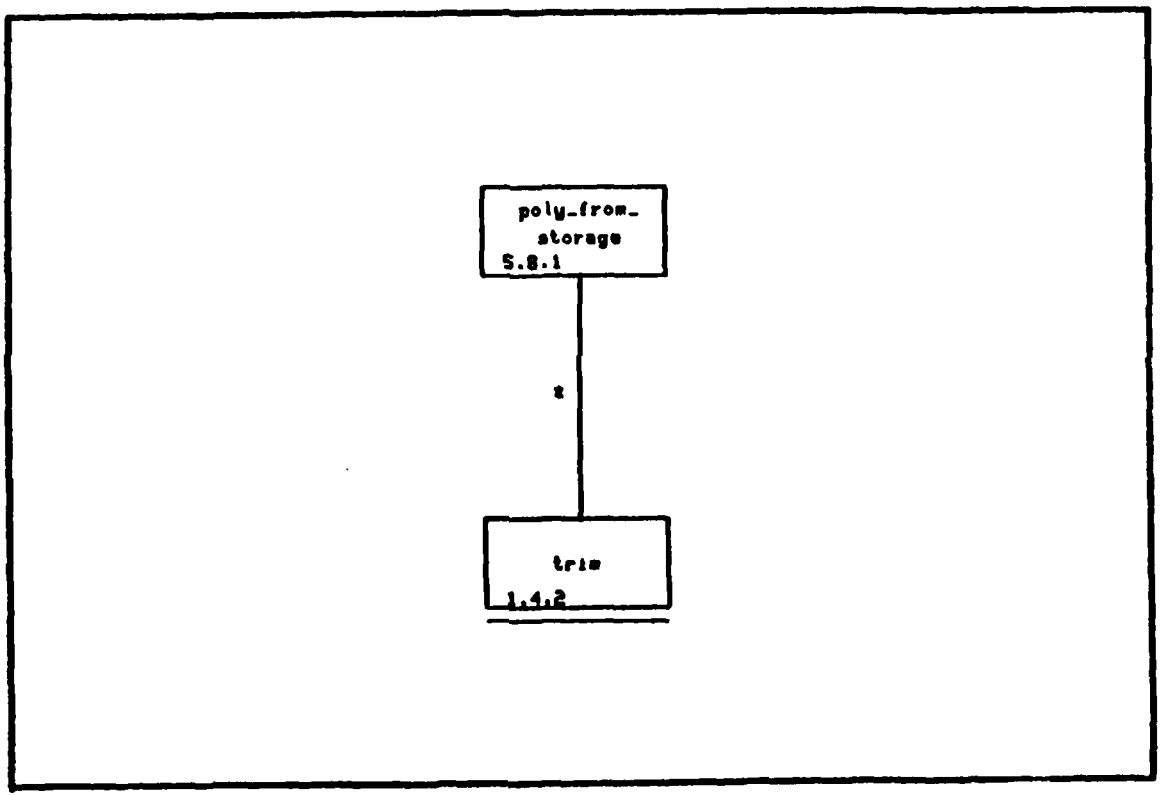

update
5.7
*
*
*
clear
1.10
goloxy
1.1
disp_msg
1.9.1
get_strng
1.2.1
NOTE: This chart is continued on the next page

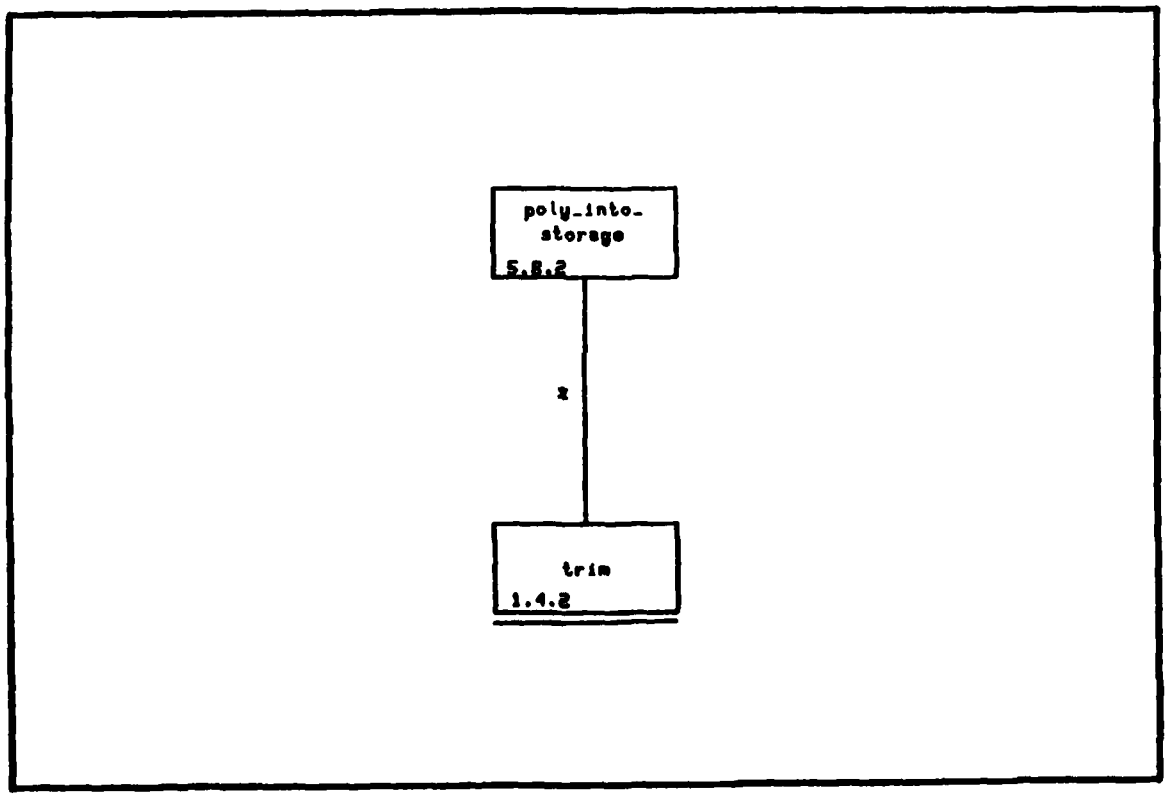

update
5.7
*
*
pause
2
clear_msg
1.9.1.2
out_string
1.5.1
NOTE: This chart is continued from previous page


form
5.8
clear
1.10
gotoxy
1.1
disp_msg
1.9.1
out_string
1.5.1
get_int
5.1.1.1
NOTE: This chart is continued on the next page


form
5.8
x
x
polymlt
5.3.2.2.1
pause
8
clear_msg
1.9.1.8
highlight
1.7.1
nohighlight
1.7.2
NOTE: This chart is continued on the next page


form
5.8
1
1
2
2
2
poly_from_
storage
5.8.1
poly_into_
storage
5.8.2
spolymlt
5.3.2.3.1
polyadd
5.3.2.2.2
diaplf
5.1.1.3
1. choice, pol
NOTE: This chart is continued from previous page


poly_from_
storage
5.8.1
*
trim
1.4.2


poly_into_
storage
5.8.2
2
trim
1.4.2

AD-A103940

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/EE/85D-46	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson, AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO.; PROJECT NO.; TASK NO.; WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): Susan K. Mashiko, B.S.A.E., Capt, USAF; Gary C. Tarczynski, B.S.E.E., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr, Mo, Day): 1985 December; 15. PAGE COUNT: 647
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES: FIELD 09, GROUP 02, SUB GR	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): MICROCOMPUTERS, COMPUTER AIDED DESIGN, COMPUTER PROGRAMS, CONTROL SYSTEMS, CONTROL SYSTEM DESIGN AND ANALYSIS, CONTROL THEORY

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: Development of a Computer Aided Design Package for Control System Design and Analysis for a Personal Computer (ICECAP-PC)

Thesis Chairman: Dr. Gary B. Lamont

Approved for public release: IAW AFR 190-1.
LYNN E. WOLAVER 16 JAN 86
Dean for Research and Professional Development
Air Force Institute of Technology (AFIT)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED X SAME AS RPT ___ DTIC USERS ___	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: [illegible]	22b. TELEPHONE NUMBER (Include Area Code); 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE — SECURITY CLASSIFICATION OF THIS PAGE

This investigation developed a computer-aided design (CAD) package for control system design and analysis. The package was implemented on different varieties of small personal computers.

Structured design and other software engineering techniques were applied during the development effort. The program consists of a keyword-driven menu structure and a set of control system analysis procedures. The analysis procedures allow input of systems which are defined by transfer functions, polynomials, and matrices. Polynomials and matrices can be manipulated mathematically, and some block diagram manipulation can performed on transfer functions as well.

The program is only part of a continuing development effort in the Information Sciences Laboratory at the Air Force Institute of Technology.

END

FILMED

3-86

DTIC